Application of Homomorphic Encryption on Neural Network in Prediction of Acute Lymphoid Leukemia

Ishfaque Qamar Khilji[1], Kamonashish Saha[2], Jushan Amin Shonon[3], Muhammad Iqbal Hossain[4]
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh

***Abstract*—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. Accurate Diagnosis of cancer is essential in treating it properly. Medical tests regarding cancer in recent times are quite expensive and not available in many parts of the world. CryptoNets, on the other hand, is an exhibit of the use of Neural-Networks over data encrypted with Homomorphic Encryption. This project demonstrates the use of Homomorphic Encryption for outsourcing neural-network predictions in case of Acute Lymphoid Leukemia (ALL). By using CryptoNets, the patients or doctors in need of the service can encrypt their data using Homomorphic Encryption and send only the encrypted message to the service provider (hospital or model owner). Since Homomorphic Encryptions allow the provider to operate on the data while it is encrypted, the provider can make predictions using a pre-trained Neural-Network while the data remains encrypted all throughout the process and finally sending the prediction to the user who can decrypt the results. During the process the service provider (hospital or the model owner) gains no knowledge about the data that was used or the result since everything is encrypted throughout the process. Our work proposes a Neural Network model which will be able to predict ALL-Acute Lymphoid Leukemia with approximate 80% accuracy using the C_NMC Challenge dataset. Prior to building our own model, we used the dataset and pre-process it using a different approach. We then ran on different machine learning and Neural Network models like VGG16, SVM, AlexNet, ResNet50 and compared the validation accuracies of these models with our own model which lastly gives better accuracy than the rest of the models used. We then use our own pre-trained Neural Network to make predictions using CryptoNets. We were able to achieve an encrypted prediction of about 78% which is close to what we achieved when validating our own CNN model that has a validation accuracy of 80% for prediction of Acute Lymphoid Leukemia (ALL).**

Keywords*—CryptoNets; neural network; Acute Lymphoid Leukemia (ALL); homomorphic

I. 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 ALL (blood 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. This system can be used in case of banks, hospitals and other sectors. In our model we included Homomorphic encryption as mentioned earlier. In this system, it will allow one party to have a public key such as in hospitals where a lot of patients can send their data through the public key which will be encrypted and stored in local servers (cloud used in future works). The owner, in our case the hospital administration, lab technicians, 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 if they want to store or export their information. In the encryption process, the owner will only have the private key and will be able to decrypt the data, on the other hand, the service provider does not have any key and hence will not be able to decrypt the data and thus they won't know about the data inside or be able to get any information about the predicted data. This will provide a better privacy and will also decrease the overall cost since there is only one private key.

Existing works of running machine learning models on encrypted data include Grapel et al. [7], where they propose confidential algorithms for binary classification based on polynomial approximations to least-squares solutions found by a small number of gradient descent steps. They show experimental validation of the confidential machine learning pipeline and discuss the give and takes involving computational complexity, prediction accuracy and cryptographic security. Zhan et al. [8] works say that their paper considers how to conduct k-nearest neighbor classification in the following scenario: multiple parties, each having a private data set, want to collaboratively build a k-nearest neighbor classifier without disclosing their private data to each other or any other parties. They intend to develop a secure protocol for multiple parties to carry out the desired calculation. All the parties take part in the encryption and in the calculation involved in learning the k-nearest neighbor classifiers. Qi &Atallah, [9] say that they use techniques to also solve the general multi-step k-NN search, and describe a particular expression of it for the case of sequence data. The protocols and correctness evidence can be extended to cope with other privacy-preserving data mining tasks, like classification and outlier detection. Aslett et al. [10][11] propose modified algorithms in application of extreme random forests, involving a new cryptographic stochastic fraction estimator, and naïve Bayes, involving a semi-parametric model for the class decision boundary, and demonstrate how they are useful in learning while predicting from encrypted data. They also exhibit that these methods perform competitively on several different classification data sets and

provide detailed information about the calculative practicalities of these and other FHE methods.

In our project, the unencrypted data will be first used to train the neural network. The training data sets can be difficult to find for these types of projects because they always have a privacy issue and also, it's not easily available. After training and validation, our Neural Network model can be used to give secure predictions regarding ALL to the target user. The problem of such type is also called Privacy Preserving Data Mining (Agrawal & Srikant) [23]. To come to the internal concept of our project, we used CryptoNets where we use Homomorphic Encryption on our own Neural Network to secure ALL predictions. This is also a work on running Neural Network and machine learning algorithms on encrypted data but what we have done is a practical implementation of the method in finding the secure predictions of a life threatening disease which is the first of its kind in terms of applying Neural Networks and Machine learning algorithms on encrypted data.

We propose to make a Privacy Preserving Neural Network model which can predict Blood Cancer as well as maintain the privacy of the patient. In our research, 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 Acute Lymphoid Leukemia. The results are then compared amongst them. Moreover, we then 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 an 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 and HE (homomorphic encryption wrapper) is implemented on it to do computations and give predictions in a secure format. We are in 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 be encrypted and the result will only be known by the patient with the private key which will ensure privacy.

Our objectives include detection of Blood Cancer (Leukemia) from imagery test samples after proper modification in order to run on the custom 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 a privacy preserving model (CryptoNets) will be the first of its kind. The process that we introduce will pave a way for implementations in various fields. This will ensure secure lives and provide customer satisfaction.

II. Background

A. Literature Survey

1) CNN Features: Shafique and Tehsin [1] used pre-trained AlexNet and fine-tuning to classify ALL subtypes on ALL-IDB augmented with 50 private images. Rehman et al. [2] used a pre-trained AlexNet and fine-tuning to classify ALL subtypes on a private dataset of 330 images. On the other hand, Vogado et al [3] used different pre-trained CNNs as fixed feature extractors to classify ALL on ALL-IDB. Amongst all these, the most informative ones are selected using PCA and classification is performed with an ensemble of MLP, random forest and SVM.

2) Handcrafted Features: Mohapatra et al. [4] and Madhloom et al. [5] use private dataset 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 is small compared to others and also tough to compare the results. The private datasets are unavailable and the public ALL-IDB datasets 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 created is done on C#. Grapel et al. [7] suggested a use of homomorphic encryption for machine learning algorithms where they focused on finding the algorithms where training can be done over encrypted data and hence were forced to use a learning algorithm where the training algorithm can be expressed 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.

B. Homomorphic Encryption

Homomorphic encryption algorithms that require one operation, such as addition, have been known for decades, such as for the ones based on the RSA or Elgamal cryptosystems. But a homomorphic encryption method that allows an infinite number of two operations, i.e. addition and multiplication, allows the computation of any circuit and thus a complete solution of homomorphic (FHE) is gained. FHE was first presented in Gentry [12]. 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 [13][14]. 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 implemented tools such as Noise Growth Simulator and Automatic Parameter Selection Module to help the user to achieve maximum performance [15]. 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 lack full homomorphic capacity. The fully homomorphic encryption supported an arbitrary number of multiplications and additions, and hence compute any form of function on encrypted information. For all forms of computations on the information warehoused in the cloud, FHE must be embraced because it allows execution of operations on encrypted records without decryption. As such, the usage of FHE is a crucial step in enhancing cloud-computing security.

C. Encoding

As described above, there is a discrepancy between the atomic structures in neural networks (real numbers) and the atomic structures in the homomorphic encryption schemes (polynomials in R^n_t) [16]. An encoding scheme will map each other in a manner that preserves 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 fixed precision numbers, and then their binary representation can be used to convert them into a polynomial with the binary expansion coefficients. This polynomial will have the property of returning the encoded value when evaluated at value 2. Another alternative is to encode as a constant polynomial the fixed number of precisions. This encoding is simple, but in the sense that only one polynomial coefficient is being used may seem inefficient. One problem with the scalar encoding is that when homomorphic operations are performed, the only coefficient of the message polynomials grows very rapidly.

D. 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**. These forces us to pick large values for t, which allows the noise to grow faster in the cipher texts 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). The concept of using multiple primes is $t_1 \ldots t_k$; 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 such polynomial is encrypted and manipulated identically. The CRT guarantees that we will be able to decode back the result, as long as its coefficient does not grow beyond $\prod t_j$. Therefore, this method allows us to encode exponentially large numbers while increasing time and space linearly in the number of primes used.

E. Plaintext 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** [15]. 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 the 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, in case 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**. In case 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**.

F. 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 details 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 of the coefficients of ciphertext polynomials. SEAL represents q as coeff modulus.
- **t,** the plaintext modulus, is an integer modulus taken in reduction of the coefficients of plaintext polynomials. SEAL shows t, as plain modulus.
- Integer coefficients are decomposed into smaller parts according to the integer base w. The integer calculates the number *w,q:=blogw(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 *log2w*. SEAL shows *log2w* 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.

G. Algorithms used

The encryption scheme we use is a public-key, homomorphic encryption scheme, and consists of the following algorithms [15]:

- A key generation algorithm **KeyGen (parms)** that, on input the system parameters "parms", generates a public/private key pair **(pk; sk)** and a public evaluation **key, evk**, which is used during homomorphic multiplication.
- An encryption algorithm **Enc(pk;m)**, that encrypts a plaintext , m, using the public **key, pk**
- A decryption algorithm **Dec (sk; c),** that decrypts a cipher text, c, with the private **key, sk**.
- A homomorphic addition operation **Add (c1; c2)** that, given as input encryptions c1 and c2 of m1 and m2, outputs a ciphertext encrypting the **sum, m1 + m2**
- A homomorphic multiplication operation **Mult (c1; c2)** that, given encryptions c1 and c2 of m1 and m2, outputs a ciphertext encrypting the **product, m1. m2**

H. Neural Network Models used

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 all are incorporated with 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.

On pre-trained CNN models as well as SVM (Support vector machine) models of our own. The CNN models that we used include VGG16 and VGG19, AlexNet and ResNet. After running these models with the mentioned dataset, we compared the accuracies (both train and test accuracies).

1) VGG16 and 19: In VGG16 architecture, the images are passed through a sequence of convolutional layers which are of fixed size (224x224 RGB image). Thus, we use the default image size for this model in our dataset. In one of the configurations, it also utilizes a 1×1 convolution filter. The convolution stride is fixed to 1 pixel. Spatial pooling is carried out by five max-pooling layers, which follow some of the convolutional layers (not all the convolutional layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride of 2. There are three fully connected layers which have different depths in different architectures. Amongst them, the first two have 4096 channels, and the third performs 2-way classification of the Leukemia dataset and contains two channels for each individual class and the last layer is a soft-max layer. This configuration is the same in all the networks. We are using pre-trained VGG16 and VGG19 models of ImageNet dataset. Thus, in building our own VGG16 model we use the "Weights" of ImageNet. We then extract features of our dataset that are 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.

2) SVM: Supervised Vector Machine (SVM) is a supervised machine learning algorithm which divides the dataset into two classes and is mostly used for classification and regression purposes. 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 of 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. Lastly, we ensure that the SVM classifier has one hyper parameter which is a penalty parameter C of the error term.

3) AlexNet: Classifying the image is a major problem and AlexNet fixes it by taking the input image of one of 1000 different groups and generally giving output of a vector of 1000 numbers. There are two groups here instead of 1000 so an output vector of only two will be present. The sum of all output vector elements is 1. AlexNet takes an RGB image size 224x224 input picture from the preprocessed dataset. Nevertheless, unless the image is not in RGB or in grayscale, it is converted to RGB by replicating the single channel in order to get a 3 channel RGB picture. AlexNet has 3 Fully Connected Layers and 5 Convolutional Layers.

- Multiple Convolutional Kernels: Multiple convolutional kernels are also many times called filters that extract the necessary features out of an image where the single convolutional layers consist of multiple similar size kernels.
- The first two Convolutional layers: The third, fourth and fifth layers of convolution are joined directly. After the fifth convolutional layer comes an Overlapping Max Pooling layer, whose output goes through a sequence of two fully integrated layers. The second fully integrated layer feeds heuristic SoftMax labelling into two classes.
- Max Pooling layers: The depth is kept unaltered by sampling the sample's height and width. Overlapping Max Pool layers are compared to Max Pool layers, other than neighboring windows where the max is estimated to overlap. Makers of the model used to pool 3x3 size windows between opposite windows, with two steps. This overlapping complexity of pooling has

helped to lower the top -1 error rate by 0.4 percent, the top-5 error rate by 0.3 percent, compared to using non-overlapping 2x2 sized pooling windows with step 2, giving identical output dimensions.

AlexNet's use of the nonlinearity function within the layers is an important feature. Activation functions of sigmoid or Tanh functions used to be the traditional method of training a neural network model. AlexNet has displayed that deep CNNs can be trained much more rapidly using ReLU's nonlinearity feature rather than using saturated activation functions such as tanh or sigmoid. Until feeding the data into the layers and constructing the model, various techniques such as image mirroring, shuffling and random cropping of images in data augmentation to minimize overfitting. This is stated earlier in this section, in which the data set explanation is present.

In dropout, one neuron with a probability of 0.5 is removed from the network. If a neuron is lost, this does not lead to propagation which is either forward or backward. Thus, each input goes through different architecture of the network due to which the learned weight parameters are therefore more robust, and are not readily overfitted. There is no dropout during testing, and the entire network is utilized, but output is scaled by a factor of 0.5 to adjust for the neurons lost during training. Dropout raises the number of iterations required to converge by a factor of 2 but AlexNet will significantly overfit without it.

4) ResNet50: Above is a pre-trained model of the ResNet50 architecture. The model has "50" layers with weights. Residual Networks or ResNet creates networks through models known as residual models and also known as the degradation problem. Although increasing depth increases the accuracy of the network, the problem increases when the vanishing gradient arises. Another issue that occurs while training the deeper network is greater training error as it adds the layers when performing optimization on large parameter space. The architecture of ResNet is identical to that of VGGNet which has 3x3 filters. The ResNet50 model we will use is a pre-trained model trained on the dataset ImageNet.

III. Proposed Model

A. Dataset Pre-Processing and Feature Selection

In 2018, another dataset with in excess of 10,000 preparing pictures and a separate test set of ordinary B-lymphoid forerunners and threatening B-lymphoblasts has been discharged as an online test open to the general population. In 2019, it was made available for general use [24]. The enormous size of this new dataset permits to make improved classifiers dependent on profound neural systems and furthermore gives an increasingly dependable correlation of contending approaches. In this work we present our way to deal with the arrangement of sound and dangerous cells on the referenced dataset utilizing a convolutional neural system. The test dataset [17,18,19,20,21], from now on alluded to as C_NMC dataset, contains pictures of white platelets taken from 154 individual subjects, 84 of which show ALL. Table I gives a nitty gritty breakdown 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 × 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 [17]. See Fig. 1 for instance pictures from the dataset.

Table I shows 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.

Despite the fact that the dataset contains in excess of 10,000 pictures, a few information enlargement strategies can be applied to build the measure of preparing information further and improve the preparation of our convolutional neural system. Since tiny pictures are invariant to flips and turns, we perform level and vertical flips with 50% likelihood each and pivots with an edge from [0, 360] degrees picked consistently at irregular. Since convolutional neural systems with pooling tasks or walks bigger than one are not flawlessly interpretation invariant, we additionally perform arbitrary interpretations of up to 20% of each side-length in flat and vertical ways. Also, the pictures are further focus trimmed to 100 × 100 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 CryptoNet model with ease. This part is explained further in the coming section.

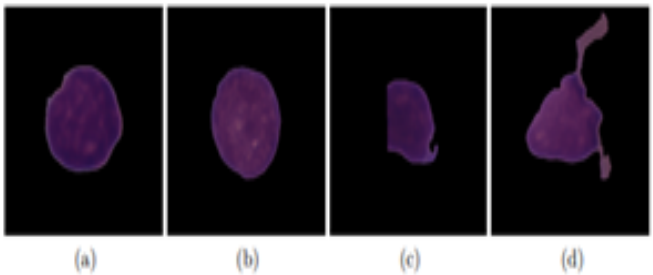

Fig 1. Images in the Training Set. (a) ALL cell (b) Normal cell (c), ALL cell with Part of the Cell Cut Off Due to an Imperfect Segmentation (d) Normal cell with Superfluous Background Due to an Imperfect Segmentation.

TABLE I. TRAIN AND TEST SUBJECTS AND THE CORRESPONDING NUMBER OF SAMPLES

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

B. Model Description

According to the workflow diagram illustrated previously in Fig. 2, firstly the C_NMC Challenge 2019 dataset is modified, pre-processed to CIFAR-10 format, split into training and test and taken in numpy arrays accordingly. The conversion of the dataset to CIFAR-10 format is essential because previously CryptoNets model has been run on mainly three datasets, namely, Cifar-10,MNIST and Caltech-101 as mentioned earlier of which Cifar-10 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 CryptoNets application using the SEAL version 3.2 HE-wrapper in C and .NET framework version 4.6.2 [16].

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 CryptoNets wrapper around it is done using code of python version 3.5.

1) Dataset Conversion and taking into Array

- After the pre-processing has been done; our 10,000 training images are at first separated equally and placed into two 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 the array would be (32,32,3). For each class subfolder, each image in the subfolder is 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. Thus the "index" value 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 into an array for each iterated image in the subfolder.
- The above steps are repeated for another class subfolder.
- 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". 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 anyway just to be on the safe side. Thus, numpy arrays are then one-hot encoded where input, that is, list of a 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.

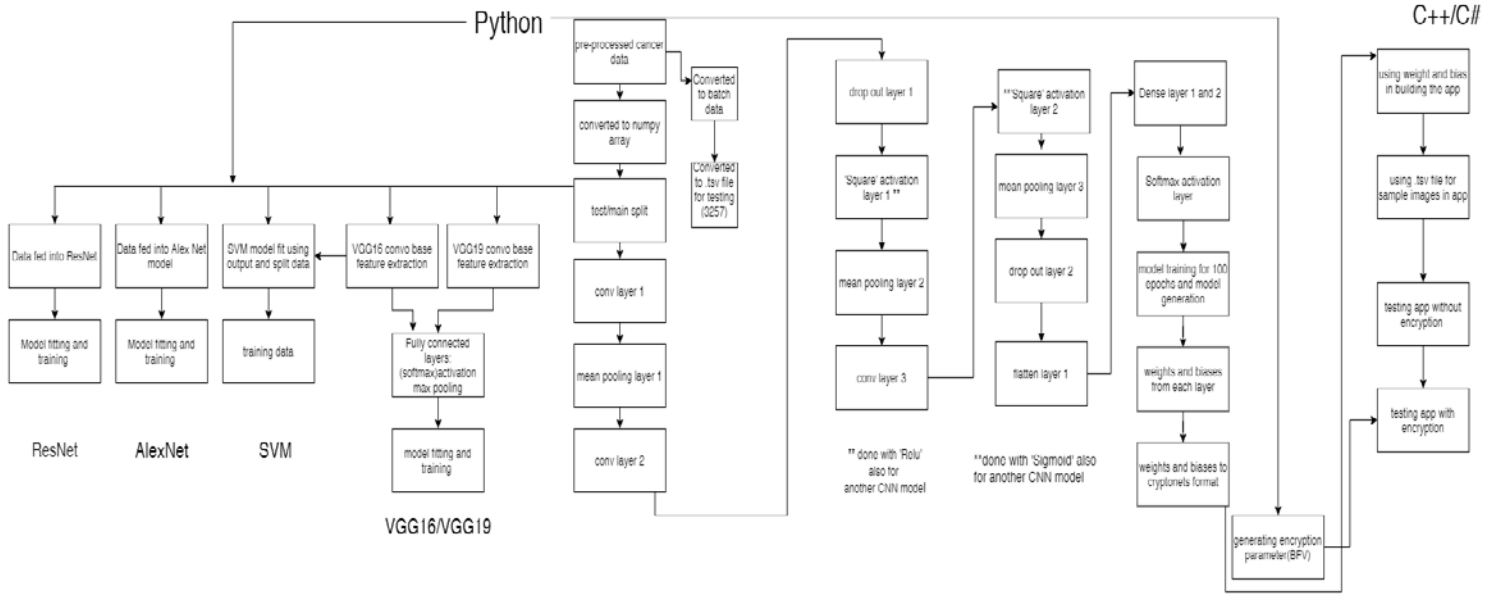

Fig 2. Overview of Proposed Approach.

2) Model Details

Our Neural Network model has 14 layers in total of which 3 are "Convolutional", 3 are "Activation", 2 "Dense" layers, 1 "flatten" layer before the output layer and the rest are "Mean Pooling" and "Dropout" layers.

The model is put into training for 100 epochs. Our own model is set to training using a different set of "Activation" layer functions twice. At first training, the first 2 "Activation" layers are "Relu" layers and the last "Activation" layer being a "Sigmoid" layer. The model is again trained this time with "Square" function instead of "Relu" and "Softmax" instead of "Sigmoid".

Below are the descriptions of the "Activation" functions mentioned.

Sigmoid: Take the value of one of the nodes in the feeding layer and evaluate the function

$z \mapsto 1/(1+\exp(-z))$

Rectified Linear: Take the value of one of the nodes in the feeding layer and compute the function

$z \mapsto \max(0,z)$

Square Activation Layer: This layer squares the value at each input node.

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

"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 possessing 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 the majority of their study to the square activation function [22]. 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 validation accuracies for both the times are recorded. For the first time the accuracy is recorded to be 78% and the second time it is recorded to be 80%.

3) Converting Weights and Biases to CryptoNets Format

Once the model is training the next step is to convert the weights and bias vectors to a format that CryptoNets recognizes. CryptoNets 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 the immediate previous or next 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 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. Values in the files are now in single columns. Thus, each column in each file of all the weights and biases for each layer is 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

4) 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 CryptoNets 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 a binary version of the 3527 test samples arranged in bytes in the .bin file. 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 CryptoNets 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 a binary version of the 3527 test samples arranged in bytes in the .bin file. The ".bin" is then converted to ".tsv" file where should have one line per image where each line contains 1 + 3*2*32 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 is converted to strings when converting to "tsv". This is done using C#.

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 a 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 in release mode.

Prior to "building" the project, the line of code:

```
var Factory = new
RawFactory((ulong)batchSize);
```

is added. The use of the "RawFactory" function is explained further.

5) Selecting Encryption Parameters

The theoretical process and mathematical formulae to calculate the correct parameters are given in the previous section "Parameter Selection". To allow correctness the parameters should support large enough numbers to be processed. Much like in traditional programming where a program might fail if numbers 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 processes in the line of code:

```
Console.WriteLine("Max computed value {0}
({1})", RawMatrix.Max,
Math.Log(RawMatrix.Max) / Math.Log(2));
```

We print 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 is provided such that the product of these numbers is 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 numbers (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 2^12 to 2^15. On the other hand, a 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 in the code, 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 for the CryptoNets build:

```
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.

The following is an output for a prediction sample generated after the CryptoNets model is run is as follows:

Fig 3. Output for a Prediction Sample.

Here in Fig. 3, label "0" is correctly predicted with an accuracy of 77.934% at an inference time of 55.20 ms.

IV. 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 "overfitting". The training and validation accuracies of the models are illustrated below:

From Fig. 4, the VGG-16 model is trained for 100 epochs. The training accuracy increases at a decreasing rate whereas the validation accuracy decreases but is very much fluctuating. At 100 epochs approaching, both the accuracies tend to become constant.

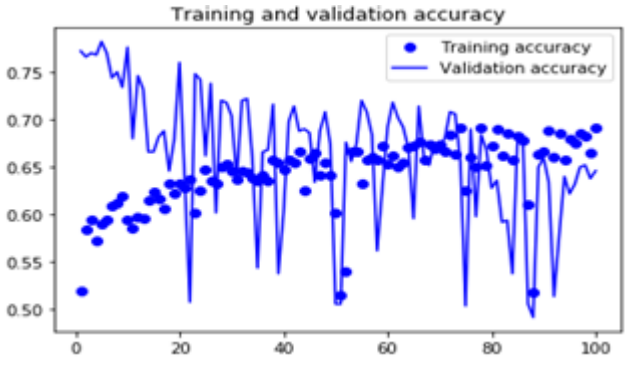

Fig 4. Training and Validation Accuracy for VGG16.

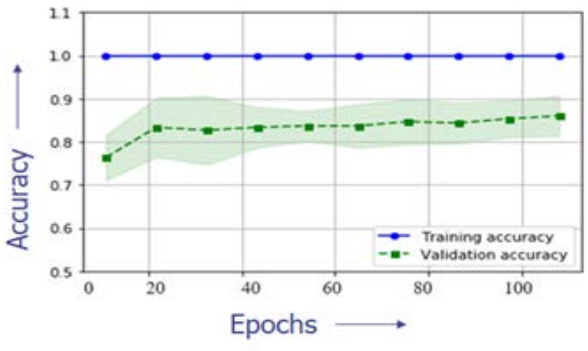

Fig 5. Training and Validation Accuracies for SVM.

It can be seen from Fig. 5, that the training accuracy is always constant at 100% which is practically unrealistic in terms of machine learning. Hence, it can be stated that this is due to overfitting of the data and we should not take this result into account.

The AlexNet model is trained for 20 epochs as depicted by the graph in Fig. 6. After 15 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.

The ResNet50 model is trained for 20 epochs. The graph depicts the Validation and Training accuracies of the model after 15 epochs Fig. 7. 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 overfitting. The model's training accuracy increases at a decreasing rate but the validation accuracy remains constant.

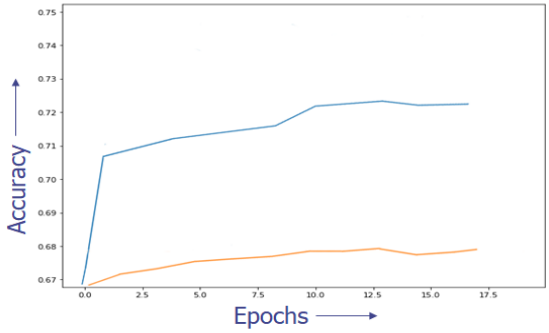

Fig 6. AlexNet Validation (Orange) and Train Accuracy (Blue).

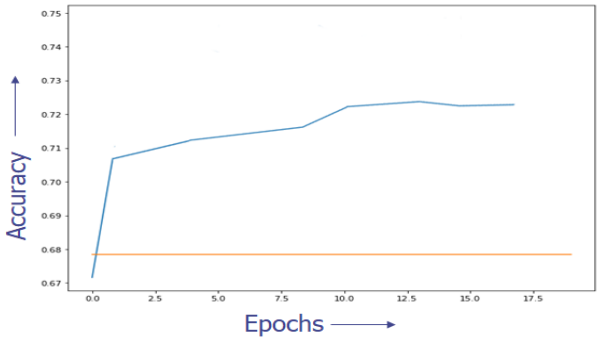

Fig 7. ResNet50 Validation (Orange) and Train Accuracy (Blue).

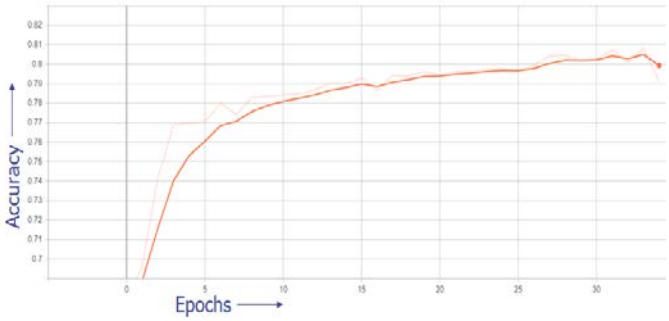

Fig 8. Validation Accuracy using Square and Softmax.

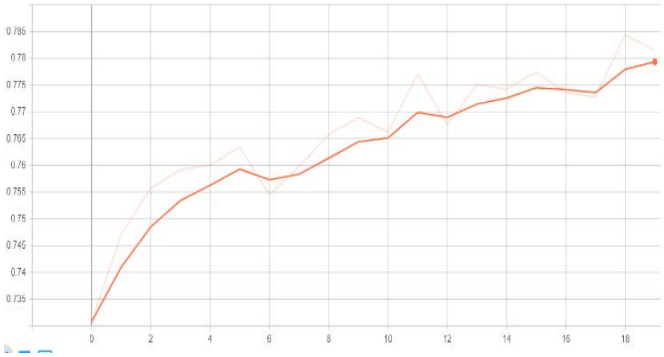

Fig 9. Validation Accuracy using Relu and Sigmoid.

Our own Neural Network model is defined as above and is trained for 100 epochs. Like mentioned earlier our model is first fitted using the "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 the "Square" function instead of "Relu" and "Softmax" instead of "Sigmoid". The graph of the validation accuracy of our own model using different sets of functions twice is illustrated in Fig. 8 and 9. The graphs were obtained from Tensorflow. Although the models with different functions are trained for different numbers of epochs, they are trained with the same dataset. Thus, there won't be much of a difference in accuracy.

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 our own model (Fig. 9).

Table II shows the comparison between all the other models.

From Table II, 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 may 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. The 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.

TABLE II. COMPARISON BETWEEN OUR 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%
Encrypted Neural Network (Accuracy)					77.934

V. Conclusion

From the earlier validation accuracies of our current model using different "Activation" functions and taking the validation accuracies of the models into account, 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. The prediction accuracy of our encrypted CNN model (77.934%) is slightly less than that of the un-encrypted CNN model (80%). This may be due to the noise generation which should reduce if correct encryption parameters are selected. In our future work, after creating the CryptoNet model, the model with 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. The cloud system does not have any key and hence will not be able to 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 secure predictions of Acute Lymphoid Leukemia (ALL) can thus be carried out through the cloud and the particular patient can access the corresponding results with ease.

According to our literature survey and our previous research, it can be seen that there are several works which used several machine learning and Neural Network algorithms in classification of Acute Lymphoid Leukemia, however our approach was different and we were able to attain a high accuracy while encrypting our dataset and using our CNN model.

Moreover, the CryptoNet model that we implemented here is currently based on The Brakerski/Fan-Vercauteren (BFV, 2012) scheme from the built in SEAL library. Our future works would also include implementing the CryptoNet model for real life applications using the faster Cheon-Kim-Kim-Song (CKKS, 2016) scheme for better accuracy in the CryptoNet model used. We are currently in the process of developing the algorithm using the CKKS scheme to precisely suit our CryptoNets model and its calculations. Also, 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. We are hopeful to successfully collect about 2000 images, label it and run it on our own CryptoNets 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.

References

[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, pp. 1-533, 2018.

[2] A. Rehman, N. Abbas, T. Saba, S. I. u. Rahman, Z. Mehmood, and H. Ko-livand, "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", 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), IEEE, pp. 367–373, 2017.

[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", International conference on advanced computer science applications and technologies (ACSAT) , IEEE, pp. 330–335, 2012.

[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), p. 363, 2013.

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

[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", The 28th International Conference on Distributed Computing Systems , IEEE, pp. 311–319, 2008.

[10] L. J. Aslett, P. M. Esperan ̧ca, and C. C. Holmes, "Encrypted statistical machine learning: New privacy preserving methods",arXiv preprint arXiv:1508.068, 2015.

[11] L. J. Aslett, P. M. Esperan ̧ca, and C. C. Holmes, "A review of homomorphic encryption and software tools for encrypted statistical machine learning",arXiv preprint arXiv:1508.06574 , 2015.

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

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

[14] 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.

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

[16] 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,pp. 201–210, 2016.

[17] 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.

[18] 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", 2018.

[19] 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, p. 82.42, 2016.

[20] 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.

[21] 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, pp. 435–443,2017.

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

[23] Agrawal, R. and Srikant, R. Privacy-preserving data mining. *ACM SIGMOD Record*, 29(2), pp.439-450, 2000.

[24] "Gupta, A., & Gupta, R. ALL Challenge dataset of ISBI 2019 [Data set]. The Cancer Imaging Archive. https://doi.org/10.7937/tcia.2019.dc64i46r",2019.