University of Windsor

A

Report

Entitled

‘A CONFIDENTIAL COVID-19 TRAINING UNDER CONVOLUTIONAL NEURAL NETWORK’

Submitted under subject

Introduction to Artificial Intelligence
(COMP-8700-1)

Prepared By:

Jay Pareshkumar Shah
Parth Hetalkumar Mistry
Reem Al-Saidi

Course Instructor
Dr. Pooya Moradian Zadeh
University of Windsor

Winter 2022

Table of Contents

List of Figures

1 Abstract

In this work, we use Rubik's cube encryption to create a secure architecture comprising a central health institution and several hospitals. The main health institution can make predictions based on the weight and encrypted patient-sensitive information. On the other hand, secure medical data processing is done so that an outside party cannot gain knowledge of the data itself. Initially, the central institution will send the base model to each hospital where training occurs. The model will be trained on-site at the hospitals. Finally, each model has its own data sets and weights for different layers. The results show that the data will remain confidential even if the accuracy decreases faster.

2 Introduction

Recently, In the Artificial intelligence (AI) field, Machine Learning (ML) has acquired an incredible deal of interest within the healthcare domain due to its performance measurement [1] [2]. Training the model and predicting with vast, massive data provide accurate and reliable results with many applications [3] [4].

In this work, we Implement COVID-19 training models while preserving confidentiality properties under conventional neural networks (CNN). We trained models in three different environments, which will represent three hospitals. The hospitals will send the weights and a subset of encrypted data to the central institution. The central institution will import the models and the encrypted data to train their model. This model will effectively predict the results.

We design a confidential architecture consisting of a central health institution and different hospitals using Rubik's cube encryption [5] The original image is scrambled using the Rubik's cube principle. Then XOR operator is applied to the scrambled image's rows and columns using two secret keys. First, the XOR operator is applied to odd rows and columns of an image using a key to create confusion. The same key is then flipped and spread even to image rows and columns.

The **motivation** for this project come to have an idea that gathers all our research interests and bridges the gap between all our work. However, we face many **challenges** during work, but we successfully deal with each of them.

First, The COVID-19 patient data is encrypted using the Tensor flow encryption TFE library [6]. Second, MATLAB software support models training and a strong mathematical cryptographic algorithmic background. Third, TensorFlow was unable to convert the encrypted models from .mat format to .h5 format even with building using ONNX. Fourth, maintaining a balance between the most two important trade-offs in this project, privacy of data and model accuracy.

To create and train the model, we used TensorFlow software, and as a data set, we used the COVID-19 Radiography Database [7], which consists of 10,192 Normal and 3,616 Covid chest X-rays.

We divided models into three parts, and each model consists of training and validation datasets. Here we used a sequential model because generally, we employ a sequential model for the simple stack of layers with precisely one input and one output tensor. We used different layers for the sequential model: 2D Convolution layer, Max pooling layer, drop out layer, flatten layer, and dense layer.

As the dataset was divided into three parts, we decided to expand our dataset by using image augmentation techniques. We used Image Data Generator Class from TensorFlow to rescale the size of chest X-ray images. After the completion of expanding the dataset, we trained our models. Therefore, the weights from each model were extracted and sent to the central institution along with a subset of encrypted images.

This will allow the privacy preservation of the patient's data. The data will not be exposed to another institution apart from the hospital. And keeping privacy in mind, the models will be trained, and a central authority can get insights and make further predictions.

3 Relevant Literature Review

Bae et al. [8] investigate the privacy concerns associated with the use of Deep Learning and neural networks. It is hampered by various regulations concerning user data privacy. They examine the evasion and poisoning attacks and defend against them using empirical and certified methods. They also consider the different potential privacy threats from service providers or users. They show that the most recent cryptographic defence methods are homomorphic encryption, secure multiparty computation, and differential privacy.

To protect the privacy of sensitive data under a convolutional neural network, Vizitiu et.al, [9] assess the privacy-preserving power of the MORE fully homomorphic encryption scheme on deep learning algorithms for personalized medicine applications without sacrificing accuracy. They suggest improving the security with more advanced homomorphic encryption and maintaining the performance in real-world applications.

Jain et.al [10] focus on securing input data and model parameters using fully homomorphic encryption FHE. They investigate three significant parameters that influence the activation function and pooling method in the ML model: depth limitation, computational precision, and polynomial degree. Investing in the right CNN design and parameter choices makes it possible to implement encrypted inference in a reasonable amount of time.

Xu et.al [11] propose a CryptoNN framework that allows neural network models to be trained over encrypted data using an emerging functional encryption scheme rather than SMC or HE. They show that CryptoNN achieves the privacy goal and model accuracy according to security testing and performance evaluation.

Gomathi et.al [12] propose a new method called ANN-based multistage image encryption using Rubik's cube method secured data transmission They implement a neural network with three layers: an input layer, a hidden layer, and an output layer. They show that ANNs can learn from their surroundings through internal parameter adaptation since it employs adaptive learning techniques.

4 Objectives

4.1 Problem Statement

Recently, Machine Learning (ML) has received a great deal of attention in the healthcare domain due to its cutting-edge performance with accurate and reliable results.

However, the adoption of Deep Learning and the neural network has been hampered by different regulations concerned about patients' health data privacy. As an example, in convolutional neural network CNN, learning models and predicting the results for COVID-19 patients will disclose patient data privacy.

In medical research cooperation, exporting results from central health institutions to other hospitals in plain text will release patients' sensitive information.

4.2 Targets

In this project, we aim to:

First, implement COVID-19 training models while preserving confidentiality property under conventional neural network (CNN).

Second, design accurate predicting models for different COVID-19 patients located at different hospitals

Third, design a confidential architecture consisting of a central health institution and different hospitals using Rubik cube encryption.

5 Project details

5.1 Requirement Specification

5.1.1 Functional requirement

We have two or more entities (which refer as hospitals) and one central authority. The central authority will have a base model and each hospital will have its own set of images divided into COVID and NORMAL images.

5.1.2 Software Requirement

The first iteration of the experiment was set up was done a machine with Intel i5 processor. The machine had 8GB of memory and 2GB of NVIDIA GeForce 940MX graphics card. Furthermore, to allow TensorFlow to use GPU, we will need to install some dependencies. These dependencies are core dependencies built by NVIDIA to have their GPUs support Tensor processing. These dependencies are the latest NVIDIA GPU Drivers, CUDA Toolkit, CUPTI, cuDNN, and respective python versions.

For further iterations, as we had a low-end machine to run the algorithm on, we needed to move to Google Colab for faster execution. Google Colab has Intel Xeon Processor @ 2.3GHz, with ~12GB of RAM, and a 12Gb of industry level GPU called TESLA K80.

5.1.3 Hardware specification

As this has a deep learning model training approach, the system should have good configurations. Deep Learning models require high computational power to manipulate the weights. Along with that, as in our approach, we use Convolutional Neural Networks, the training will require graphic processing units to perform the tasks faster. As we have images, we will require good computational power to process them. To transfer the weights and images, the hospitals should have a secure link setup with higher bandwidth to fasten up the transfer.

5.1.4 Privacy Requirement

Increasing patient trust towards their own private data by providing data encryption to their sensitive X-ray images. Second, No third party will disclose the sensitive data or results while being transmitted as it is encrypted under a confidential architecture under the assumption that this central health center is trusted. Finally, Improving health cooperation among different health institutions by making confidential data transference among several medical organizations. The construction of the architecture is based on Rubik cube encryption [8] with two secret keys.

5.2 Definitions and terminology

Machine learning is an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment [13].

The part of Machine Learning that we will use in our project is called deep learning which consists of neural network-like algorithms. A neural network consists of an input layer of neurons (or nodes, units), one or two (or even three) hidden layers of neurons, and a final layer of output neurons [14].

In TensorFlow, a neural network is a sequence of layers created using a sequential model. A Sequential model is a stack of layers where each layer has exactly one input tensor and one output tensor [15].

A layer in a deep learning model is a structure or network topology in the architecture of the model, which takes information from the previous layers and then passes information to the next layer [16].

During the training process, the algorithms will tweak the weights of the layers. Weight is the parameter within a neural network that transforms input data within the network's hidden layers [17]. We also use augmentation techniques on images to expand the dataset. Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset [18].

Rubik's cube encryption [5] The original image is scrambled using the principle of Rubik's cube. Then, the XOR operator is applied to rows and columns of the scrambled image using two secret keys. To confuse the relationship between original and encrypted images, the XOR *operator* is applied to odd rows and columns of the image using a key. The same key is flipped and applied to even rows and columns of the image.

Encryption is referred to as the process of converting data to an unrecognizable or —encrypted" form, (encoding) a message so that it can be read-only by the sender and the intended recipient [19].

Confidentiality is defined as a term used to prevent the disclosure of information to unauthorized individuals or systems [20].

Homomorphic encryption is a form of encryption that allows specific types of computations to be carried out on ciphertexts and generates an encrypted result that, when decrypted, matches the result of operations performed on the plaintexts. This is a desirable feature in modern communication system architectures [21].

5.3 Architectural design

The project starts with four entities. Three of them are hospitals and the final one is the central institution. The central institution will send the base model to each hospital where the training will take place. We use the COVID-19 Radiography Dataset[9] and we will divide it into 3 parts and will be assigned to each hospital. The hospitals will train the model locally. So, each model has its own data sets and has different weights for different layers.

After training and evaluating the weights, hospitals will send the corresponding weights to the central institution. At the same time, the hospitals will also send a subset of encrypted images to the central institution.

We utilized the iterative methodology approach to build the project. We developed the project in smaller units using the iterative technique, which results in a shorter period for model creation and is often employed in cases when time is a constraint. The central authority will now send the base model to the hospitals. This model will have a combination of layers like Conv2D and Dense layers. Each hospital will now start training its models. As the data of each hospital is different, their learning outcomes will be different.

After successfully training the model, each hospital will start encrypting the images. The images are encrypted using Rubik's Cube Encryption algorithm[5]. After the encryption, a subset of the encrypted images will be sent to the central institution.

The encryption is performed with the augmented image that apply Rubik cube principles [5]. Then, XOR operator is applied to rows and columns of the scrambled image using two secret keys. For the purpose of confusing the relationship between original and encrypted images. The same key is flipped and applied to even rows and columns of the image. See Figure 1

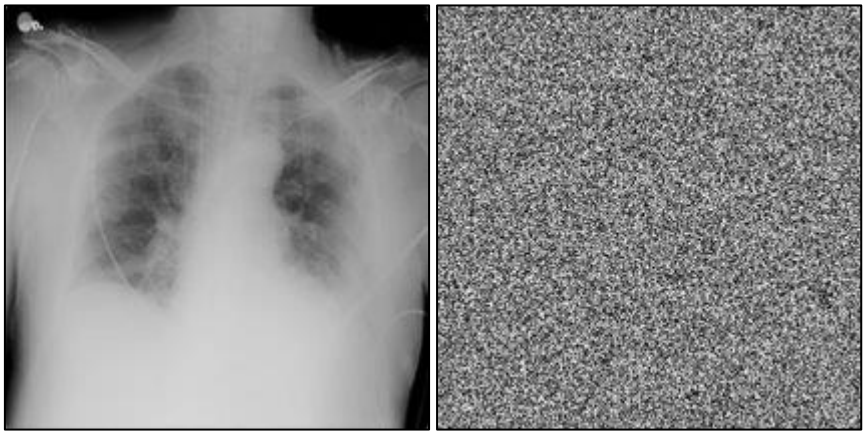


Figure 1 Image before and after encryption using Rubik cube encryption

Along with the set of encrypted images, hospitals will also send the model weights of each layer. Therefore, the central institution will have a set of a subset of encrypted images from each hospital and respective model weights.

Now, the central institution will use the weights, and the encrypted images to train the model. Therefore, the data of each hospital was accessible neither to the central institution nor to the other hospitals. This maintained privacy among the hospitals. The central institution can also work as a repository of encrypted images.

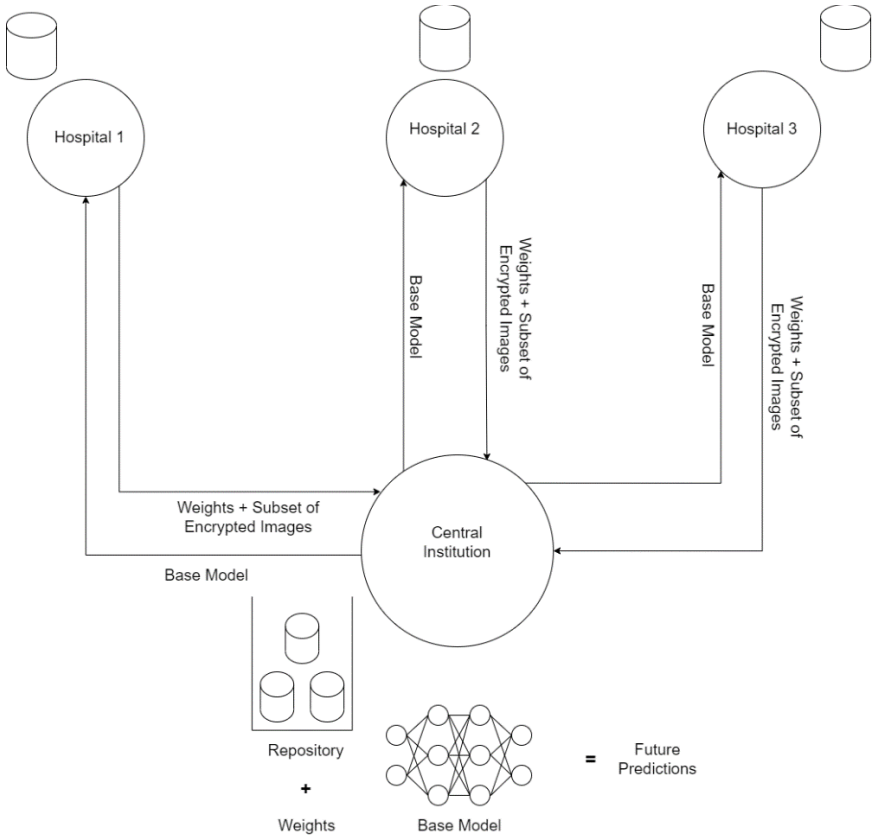


Figure 2 – System Architecture design

The architecture consists of 4 entities. Three of the entities are hospitals and one of them is a central institution. The central institution will send a base model to all the 3 institutions for training the model. Each hospital has its own confidential data which must not be shared with other hospitals. Each hospital will train the model using the raw data and will try to gain the highest accuracy. The rule is that no hospital can tweak the model. After training, the weights of the layers in the model will be extracted. Along with that, they will encrypt a subset of the images which were used during the training. The hospitals will send the subset of encrypted images and the weights to the central institution. Similarly, the other 2 hospitals would also have sent similar data, but the weights will be different as each hospital has different data. The central institution will import the average/best weights from the set of weights and will start training their own model. The final model will be used to make further predictions and for gathering insights.

The first iteration of the experiment was set up was done a machine with Intel i5 processor. The machine had 8GB of memory and 2GB of NVIDIA GeForce 940MX graphics card. Furthermore, to allow TensorFlow to use GPU, we will need to install some dependencies. These dependencies are core dependencies built by NVIDIA to have their GPUs support Tensor processing. These dependencies are the latest NVIDIA GPU Drivers, CUDA Toolkit, CUPTI, cuDNN, and respective Python versions.

For further iterations, as we had a low-end machine to run the algorithm on, we needed to move to Google Colab for faster execution. Google Colab has Intel Xeon Processor @ 2.3GHz, with ~12GB of RAM, and a 12Gb of industry level GPU called TESLA K80.

6 Experimental Setup

To run the algorithms, we will require Python of version above 3.7.3 and TensorFlow with graphical processing unit (GPU) support. To allow TensorFlow to take the advantage of the computational power of the GPU, we will need to install some general requirements which are specific dependencies of the GPU. These dependencies are the latest NVIDIA GPU Drivers, CUDA Toolkit, CUPTI, cuDNN.

These specifications are limited to GPUs built by NVIDIA. There can be another solution to the different kind of GPUs built by different companies. Furthermore, if there is no access to GPU, we can take advantage of Google Colab/Kaggle Kernels and use their hardware to get our results.

To emulate this solution, you will need to have a base model ready at the central institution side. There will be three separate environments where the model trainings and dataset augmentations will take place.

The programming language used here to run these tests are on Python. The Deep Learning framework used in this system is TensorFlow and other libraries for data manipulation are pandas, NumPy and matplotlib.

For the experiment, we took a COVID-19 Radiography Database [9], consisting of 10,192 Normal and 3,616 Covid chest X-rays. Here we used a sequential model. First divided models into three parts, and each model consists of training and validation datasets. We used different layers for the sequential model: 2D Convolution layer, max pooling layer, drop out layer, flatten layer, and dense layer. After completing the image escalation and modification, we trained our models with 8 steps per epoch and for 10 epochs, model 1, gets the validation accuracy of 0.7188. For Model-2, the validation accuracy is 0.6552, and for Model-3, the validation accuracy is 0.7114, and the accuracy before the encryption process.

Initially, when we started training the model on our machines, it took almost 1 and a half hours to train the model for 10 epochs. After that, when we switched to Google Colab, we got results faster in just 25 minutes. Also, as the memory sizes vary on a large scale, the machine was able to handle large calculations and retrieve better accuracy. The final accuracies we received on each model are 71%, 65% and 71%.

There were some challenges we faced during the experiment:

1. We were not able to find a lab to conduct experiments.
2. We were not able to convert TensorFlow models to .mat models for encryption in MATLAB
3. We tried TensorFlow's encryption library TFE, but it kept adding an extra dimension to the data and then CNN would not get trained
4. We tried creating models in MATLAB and then converting the .mat models to .h5. However, it was not possible even with using ONNX bridge

5. we tried to use an approach the that had a trade-off between privacy and accuracy

7 Results

We present the training accuracy compared with validation accuracy; training loss compared with validation loss for all the three models.

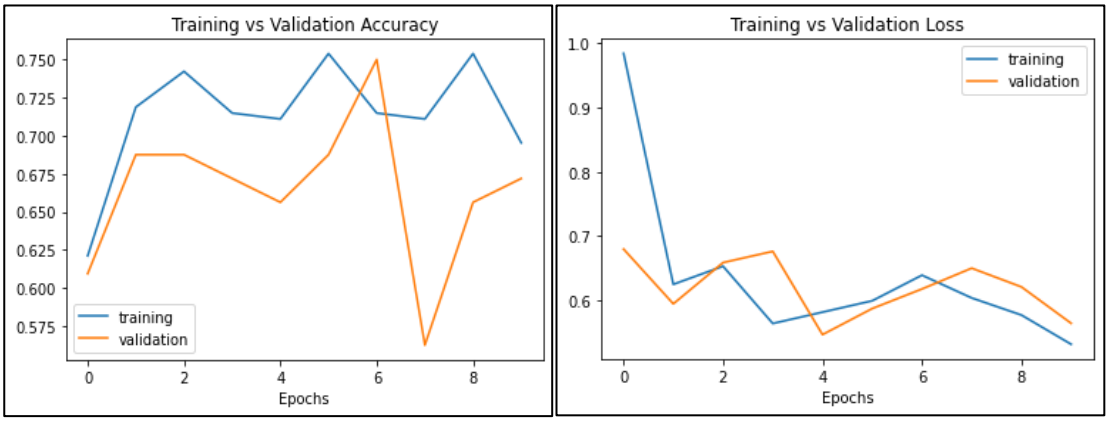


Figure 3 - Accuracy and Loss for Model 1

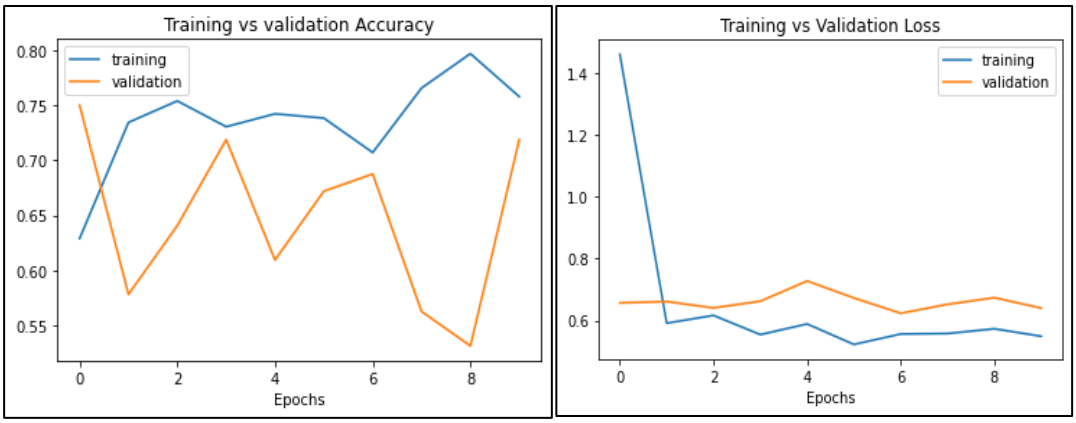


Figure 4 - Accuracy and Loss for Model 2

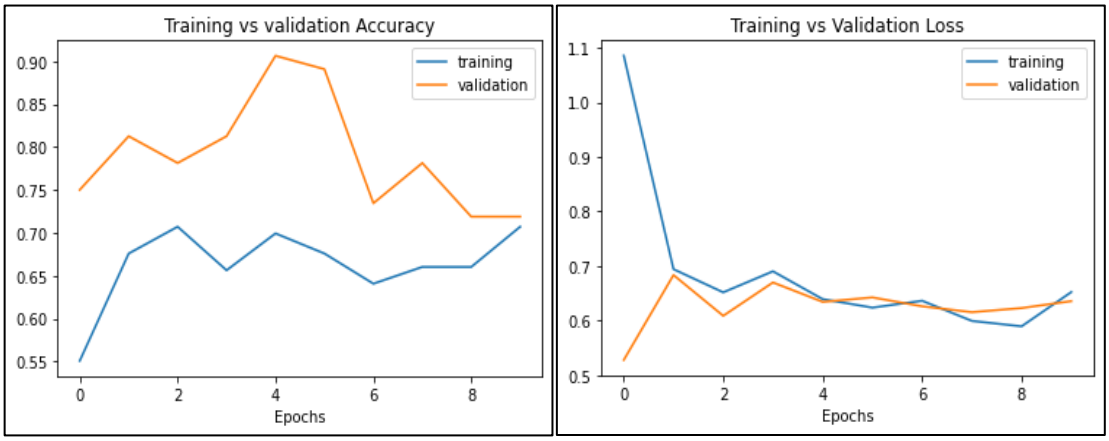


Figure 5 - Accuracy and Loss for Model 3

8 Conclusion

We can summarize, that the central health institution can perform predictions from the weight and the encrypted patient-sensitive information. However, the secure processing of medical data is done so that an external party cannot derive knowledge about the data itself. The results show that the data will remain confidential even if the accuracy is getting lower than expected since a tradeoff between privacy and accuracy was our concerns in this research project.

9 Future work

We suggested to extend this work to Add more neural network layers to train the models with more than three models and gain a balance between accuracy and privacy. As well, Implementing different privacy-preserving techniques with secure multiparty computation among different health institutions under a cloud environment.

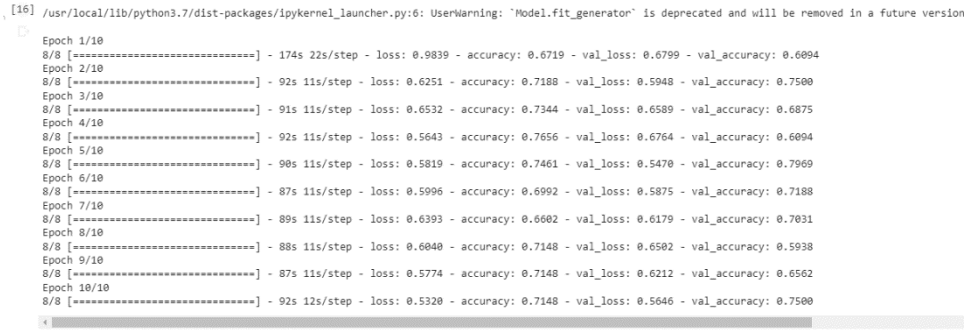
10 References

[1] Khoa, N. L. D., Makki Alamdari, M., Rakotoarivelo, T., Anaissi, A., & Wang, Y. (2018). Structural health monitoring using machine learning techniques and domain knowledge based features. In Human and machine learning (pp. 409-435). Springer, Cham.
[2] Sasubilli, S. M., Kumar, A., & Dutt, V. (2020, June). Machine learning implementation on medical domain to identify disease insights using TMS. In 2020 International Conference on Advances in Computing and Communication Engineering (ICACCE) (pp. 1-4). IEEE.
[3] Vellido, A. (2020). The importance of interpretability and visualization in machine learning for applications in medicine and health care. Neural computing and applications, 32(24), 18069-18083.
[4] Bochare, A., Gangopadhyay, A., Yesha, Y., Joshi, A., Yesha, Y., Brady, M., ... & Rishe, N. (2014). Integrating domain knowledge in supervised machine learning to assess the risk of breast cancer. International Journal of Medical Engineering and Informatics, 6(2), 87-99.
[5] Loukhaoukha, Khaled & Chouinard, Jean-Yves & Abdellah, Berdai. (2012). A Secure Image Encryption Algorithm Based on Rubik's Cube Principle. Journal of Electrical and Computer Engineering. 10.1155/2012/173931.
[6] T. (z.d.). GitHub - tf-encrypted/tf-encrypted: A Framework for Encrypted Machine Learning in TensorFlow. GitHub. Geraadpleegd op 14 april 2022, van https://github.com/tf-encrypted/tf-encrypted
[7] Rahman, T. (2022, March 19). Covid-19 radiography database. Kaggle. Retrieved April 11, 2022, from https://www.kaggle.com/tawsifurrahman/covid19-radiography-database
[8] Bae, H., Jang, J., Jung, D., Jang, H., Ha, H., Lee, H., & Yoon, S. (2018). Security and privacy issues in deep learning. arXiv preprint arXiv:1807.11655.
[9] Vizitiu, A., Niţă, C. I., Puiu, A., Suciu, C., & Itu, L. M. (2019, June). Towards privacy-preserving deep learning based medical imaging applications. In 2019 IEEE international symposium on medical measurements and applications (MeMeA) (pp. 1-6). IEEE.
[10] Jain, N., Nandakumar, K., Ratha, N., Pankanti, S., & Kumar, U. (2021). Efficient cnn building blocks for encrypted data. arXiv preprint arXiv:2102.00319.
[11] Xu, R., Joshi, J. B., & Li, C. (2019, July). Cryptonn: Training neural networks over encrypted data. In 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS) (pp. 1199-1209). IEEE.
[12] Gomathi, T., & Shivakumar, B. L. (2016). A secure image encryption algorithm based on ANN and Rubik's cube principle. ARPN J. Eng. Appl. Sci, 11(1).
[13] El Naqa, I., Murphy, M.J. (2015). What Is Machine Learning?. In: El Naqa, I., Li, R., Murphy, M. (eds) Machine Learning in Radiation Oncology. Springer, Cham. https://doi.org/10.1007/978-3-319-18305-3_1
[14] Wang, SC. (2003). Artificial Neural Network. In: Interdisciplinary Computing in Java Programming. The Springer International Series in Engineering and Computer Science, vol 743. Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-0377-4_5
[15] The Sequential model | TensorFlow Core. (n.d.). TensorFlow. Retrieved April 14, 2022, from https://www.tensorflow.org/guide/keras/sequential_model
[16] Wikipedia contributors. (2021, May 5). Layer (deep learning). Wikipedia. https://en.wikipedia.org/wiki/Layer_(deep_learning)
[17] DeepAI. (2020, June 25). Weight (Artificial Neural Network). https://deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network
[18] Brownlee, J. (2019, July 5). How to Configure Image Data Augmentation in Keras. Machine Learning Mastery. Retrieved April 14, 2022, from https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[19] Burkert, H. (1997). Privacy-enhancing technologies: Typology, critique, vision. Technology and privacy: The new landscape, 125.
[20] Sattarova Feruza, Y., & Kim, T. H. (2007). IT security review: Privacy, protection, access control, assurance and system security. International journal of multimedia and ubiquitous engineering, 2(2), 17-32.
[21] Yi, X., Paulet, R., & Bertino, E. (2014). Homomorphic encryption. In Homomorphic encryption and applications (pp. 27-46). Springer, Cham.

Appendix A

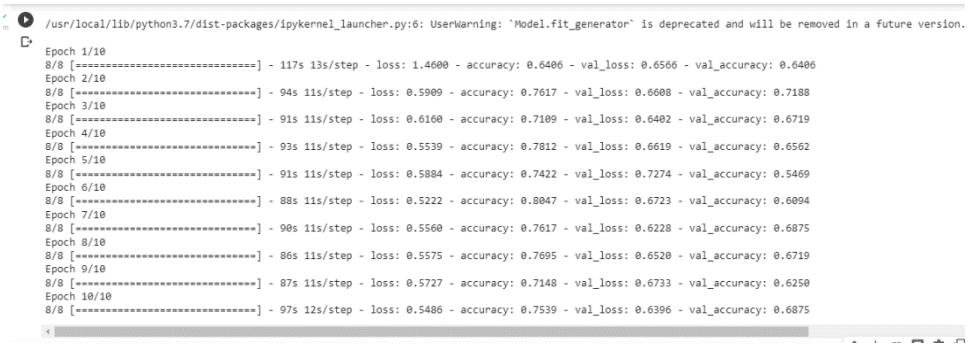
Implementation and training logs



```
[16] /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version

Epoch 1/10
8/8 [==============================] - 174s 22s/step - loss: 0.9839 - accuracy: 0.6719 - val_loss: 0.6799 - val_accuracy: 0.6094
Epoch 2/10
8/8 [==============================] - 92s 11s/step - loss: 0.6251 - accuracy: 0.7188 - val_loss: 0.5948 - val_accuracy: 0.7500
Epoch 3/10
8/8 [==============================] - 91s 11s/step - loss: 0.6532 - accuracy: 0.7344 - val_loss: 0.6589 - val_accuracy: 0.6875
Epoch 4/10
8/8 [==============================] - 92s 11s/step - loss: 0.5643 - accuracy: 0.7656 - val_loss: 0.6764 - val_accuracy: 0.6094
Epoch 5/10
8/8 [==============================] - 90s 11s/step - loss: 0.5819 - accuracy: 0.7461 - val_loss: 0.5470 - val_accuracy: 0.7969
Epoch 6/10
8/8 [==============================] - 87s 11s/step - loss: 0.5996 - accuracy: 0.6992 - val_loss: 0.5875 - val_accuracy: 0.7188
Epoch 7/10
8/8 [==============================] - 89s 11s/step - loss: 0.6393 - accuracy: 0.6602 - val_loss: 0.6179 - val_accuracy: 0.7031
Epoch 8/10
8/8 [==============================] - 88s 11s/step - loss: 0.6040 - accuracy: 0.7148 - val_loss: 0.6502 - val_accuracy: 0.5938
Epoch 9/10
8/8 [==============================] - 87s 11s/step - loss: 0.5774 - accuracy: 0.7148 - val_loss: 0.6212 - val_accuracy: 0.6562
Epoch 10/10
8/8 [==============================] - 92s 12s/step - loss: 0.5320 - accuracy: 0.7148 - val_loss: 0.5646 - val_accuracy: 0.7500
```

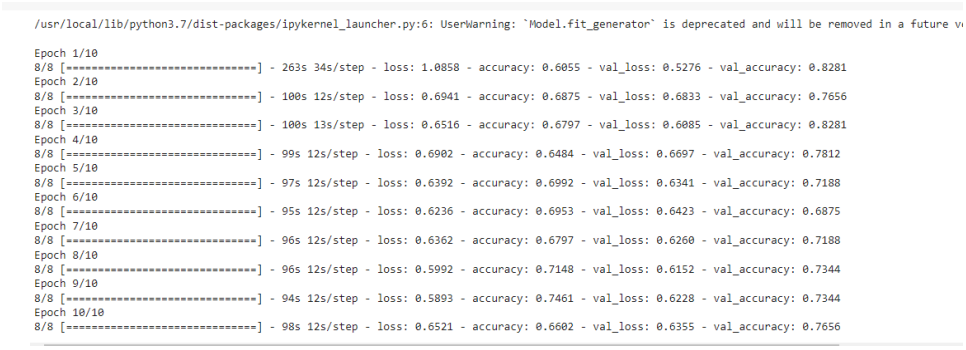
Figure 6 - Logs for Model 1



```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version.

Epoch 1/10
8/8 [==============================] - 117s 13s/step - loss: 1.4600 - accuracy: 0.6406 - val_loss: 0.6566 - val_accuracy: 0.6406
Epoch 2/10
8/8 [==============================] - 94s 11s/step - loss: 0.5909 - accuracy: 0.7617 - val_loss: 0.6608 - val_accuracy: 0.7188
Epoch 3/10
8/8 [==============================] - 91s 11s/step - loss: 0.6160 - accuracy: 0.7109 - val_loss: 0.6402 - val_accuracy: 0.6719
Epoch 4/10
8/8 [==============================] - 93s 11s/step - loss: 0.5539 - accuracy: 0.7812 - val_loss: 0.6619 - val_accuracy: 0.6562
Epoch 5/10
8/8 [==============================] - 91s 11s/step - loss: 0.5884 - accuracy: 0.7422 - val_loss: 0.7274 - val_accuracy: 0.5469
Epoch 6/10
8/8 [==============================] - 88s 11s/step - loss: 0.5222 - accuracy: 0.8047 - val_loss: 0.6723 - val_accuracy: 0.6094
Epoch 7/10
8/8 [==============================] - 90s 11s/step - loss: 0.5560 - accuracy: 0.7617 - val_loss: 0.6228 - val_accuracy: 0.6875
Epoch 8/10
8/8 [==============================] - 86s 11s/step - loss: 0.5575 - accuracy: 0.7695 - val_loss: 0.6520 - val_accuracy: 0.6719
Epoch 9/10
8/8 [==============================] - 87s 11s/step - loss: 0.5727 - accuracy: 0.7148 - val_loss: 0.6733 - val_accuracy: 0.6250
Epoch 10/10
8/8 [==============================] - 97s 12s/step - loss: 0.5486 - accuracy: 0.7539 - val_loss: 0.6396 - val_accuracy: 0.6875
```

Figure 7 - Logs for Model 2



```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future v

Epoch 1/10
8/8 [==============================] - 263s 34s/step - loss: 1.0858 - accuracy: 0.6055 - val_loss: 0.5276 - val_accuracy: 0.8281
Epoch 2/10
8/8 [==============================] - 100s 12s/step - loss: 0.6941 - accuracy: 0.6875 - val_loss: 0.6833 - val_accuracy: 0.7656
Epoch 3/10
8/8 [==============================] - 100s 13s/step - loss: 0.6516 - accuracy: 0.6797 - val_loss: 0.6085 - val_accuracy: 0.8281
Epoch 4/10
8/8 [==============================] - 99s 12s/step - loss: 0.6902 - accuracy: 0.6484 - val_loss: 0.6697 - val_accuracy: 0.7812
Epoch 5/10
8/8 [==============================] - 97s 12s/step - loss: 0.6392 - accuracy: 0.6992 - val_loss: 0.6341 - val_accuracy: 0.7188
Epoch 6/10
8/8 [==============================] - 95s 12s/step - loss: 0.6236 - accuracy: 0.6953 - val_loss: 0.6423 - val_accuracy: 0.6875
Epoch 7/10
8/8 [==============================] - 96s 12s/step - loss: 0.6362 - accuracy: 0.6797 - val_loss: 0.6260 - val_accuracy: 0.7188
Epoch 8/10
8/8 [==============================] - 96s 12s/step - loss: 0.5992 - accuracy: 0.7148 - val_loss: 0.6152 - val_accuracy: 0.7344
Epoch 9/10
8/8 [==============================] - 94s 12s/step - loss: 0.5893 - accuracy: 0.7461 - val_loss: 0.6228 - val_accuracy: 0.7344
Epoch 10/10
8/8 [==============================] - 98s 12s/step - loss: 0.6521 - accuracy: 0.6602 - val_loss: 0.6355 - val_accuracy: 0.7656
```

Figure 8 - Logs for Model 3

Appendix B

1. https://github.com/dattasiddhartha/Encrypted-machine-learning-with-simple-homomorphic-encryption/blob/master/model.py
2. https://github.com/piyush2896/Encrypted-Deep-Learning/blob/master/Syft%20-%20Paillier%20Encrypted%20Linear%20Classification.ipynb
3. https://github.com/sahibjotsingh/Image-Encryption-Based-on-Rubiks-Cube
4. https://github.com/fentec-project/neural-network-on-encrypted-data
5. https://github.com/tf-encrypted/tf-encrypted/issues/106