Privacy-preserving Machine Learning using Secure Multiparty Computation for Medical Image classification

Shreyansh Singh - 16075052
BTP Guide - Prof. K.K. Shukla

Aim - To provide private predictions from a Deep Learning model in a Medical Image classification setting

Steps Involved

- Model training on a public dataset
- Secure and serve the deep learning model
- Query the secured model to receive private predictions

Secure Multiparty Computation

Secure Multiparty Computation is a sub-field of cryptography where the goal is to create a provision for parties to jointly compute a function over their inputs which are kept private i.e. not shared with the other parties. Theoretically, the definition of MPC involves defining -

- Functionality
- Security type
- Adversarial model
- Network model

Security type

- **Computational (cryptographic) security** - Computational security asks only that no polynomial-time adversary can tell which of the messages that could potentially be plaintexts corresponding to a ciphertext is more likely than the other to be the actual plaintext. In other words, the key space is big enough to make brute-force attacks impossible for such an adversary.
- **Statistical Security** - A slightly weaker notion of perfect security, where the adversary is given a small advantage (usually denoted by ϵ) in breaking the security of the algorithm than a purely random guess.
- **Perfect security** - An algorithm is perfectly-secure if given a ciphertext, every message in the message space is exactly as likely to be the plaintext, i.e., the plaintext is independent of the ciphertext. This implies that even a computationally-unbounded adversary cannot learn anything about the plaintext.

Adversarial Model

The Adversarial model can be described in different ways -

- Adversarial Behavior
  - Semi honest
  - Fail stop
  - Malicious
- Adversarial Power
  - Polynomial time
  - Computationally unbounded

Adversarial Model (contd.)

- Adversarial Corruption
  - Static
  - Adaptive
  - Mobile
- Number of corrupted parties
  - No honest majority
  - Honest majority $t < n/2$
  - Two-thirds majority $t < n/3$

SMPC gives a combination of encryption, distribution and distributed combination and this has a big impact on data security and data privacy.

SMC - Two-party computation

- First introduced by Yao in 1986.
- Yao’s garbled circuits facilitates two-party secure computation in which two mistrusting parties can jointly evaluate a function over their private inputs without the presence of a trusted third party.
- Yao’s basic protocol is secure against semi-honest adversaries.
- It was a bit late, in 2007, that 2PC protocols for malicious setting (secure against active adversaries) were proposed.

SMC - Multi-party computation

- Secret sharing forms the fundamentals of MPC. The two most commonly used methods are Shamir's secret sharing and additive secret sharing.
- Currently, one of the most common MPC protocols used is SPDZ. SPDZ uses additive secret shares and is secure against active adversaries (malicious, dishonest majority).
- Other popular frameworks include Obliv-C, ABY, SCALE-MAMBA and FRESCO.

  These protocols are suited for general MPC but not for integration with Machine Learning.

- Some SMC frameworks that focus on ML applications are - SecureML, GAZELLE and ABY3.

Differential Privacy

- Differential Privacy (DP) is a rigorous mathematical framework which allows sharing information about a dataset publicly by describing the patterns of groups of the dataset without revealing information about the individuals in the dataset.
- An algorithm is said to be differentially private if and only if the inclusion of any one instance in the training dataset causes only statistically minor changes to the output of the algorithm.
- The role of DP here is to limit the attacker’s ability to infer such membership by putting a theoretical limit on the influence that a single individual can have.

Differential Privacy - Formal definition

A randomized mechanism K provides (ϵ, δ) differential privacy if for any two neighboring database D_1 and D_2 that differ in only a single entry, $\forall S \subseteq Range(K)$

$$\Pr(K(D_1) \in S) \leq e^{\epsilon} \Pr(K(D_2) \in S) + \delta$$

If δ = 0, K is said to satisfy ϵ -differential privacy.

The idea is that to achieve DP, noise is added to the algorithm's output. This noise is depends on the sensitivity of the output, where sensitivity is the measure of the maximum change of output due to the inclusion of a single data instance.

Security vs Accuracy tradeoff

Using DP brings in a tradeoff between privacy and accuracy of the model.

- The aim of DP is to minimise the "information leak" from a single query, but keeping this value small enough when multiple queries are made can become a challenge as for every query, the total “information leak” will increase.
- As a solution, more noise has to be injected in the data to minimise the privacy leakage but that would mean the accuracy of the model will go down.

Model Serving

- We use Secure-NN as the underlying protocol for the Secure Multiparty Computation implementation.
- Secure-NN was developed by Microsoft Research, as a 3-party SMC protocol especially for Neural Networks.
- For model serving, we set up three local servers.
- The idea is to split the model weights into shares, then send a share of each value to the different servers.
- The key property is that if we look at the share on one server, it reveals nothing about the original value (input data or model weights).
- After the model is encrypted and the weights are shared we have to set up a QueueServer to serve our model and the model is ready to accept incoming prediction requests.

Private Prediction for the Client

- After encrypting and hosting the model, we have to query the model
- This will also make use of the three local servers set up by us.
- Just like the model weights, the data is also split into shares, then send a share of each value to the different servers.
- We load the configuration file created while setting up/hosting the model.
- After this, we can provide raw data as inputs to the model (in the required shape) and the model can respond to our queries.

The entire process will be secure enough so that any outsider can not figure out the data being sent to the mode. Even the model would not know about the data that it is working on, since it is now encrypted in a way that it can work only on shares of data.

Data Collection and Analysis

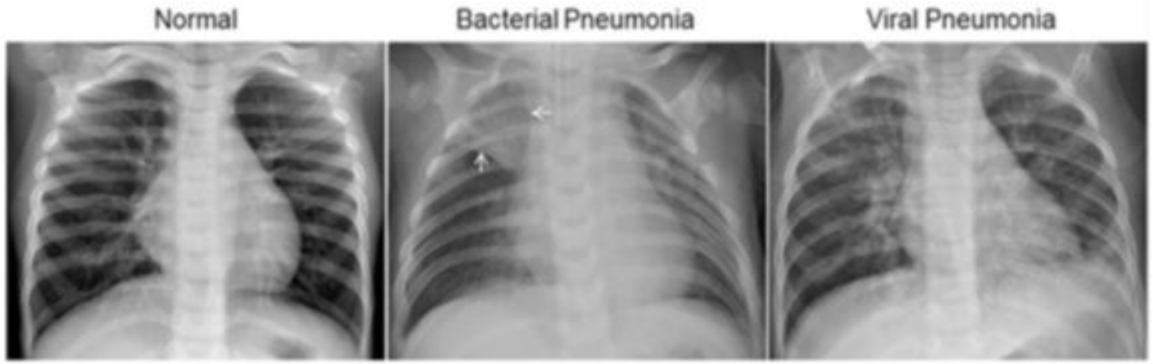
Dataset

Our research takes up the task of detecting pneumonia in patients by analysing their chest X-ray images.

This dataset was collected as a part of a research published in the scientific journal *Cell* in 2018, where the authors have collected such medical images and aim to apply image-based deep learning to detect such diseases.

A copy of the dataset is also available on Kaggle.

Dataset (contd.)



Above figure shows the variation in the x-rays for the different kinds of pneumonia. According to

- Normal chest X-ray (left) depicts clear lungs without any areas of abnormal opacification in the image.
- Bacterial pneumonia (middle) typically exhibits a focal lobar consolidation, in this case in the right upper lobe (white arrows)
- Viral pneumonia (right) manifests with a more diffuse “interstitial” pattern in both lungs.

Description of the dataset

- All the chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training any AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.
- We convert this problem into a two class classification problem because of the imbalance in the dataset, chest x-rays depicting Pneumonia and Normal images.
- In the training data, we have **1341 normal images** and **3882 pneumonia** images.
- In the test data, we have **234 normal** images and **390 pneumonia** images.

Data Preprocessing

For data preprocessing two different approaches were taken -

- Resizing images into (125, 150) using the Pillow library in Python
- Resize the image into size (224, 224) using cv2.INTER_CUBIC, i.e. cubic interpolation using the OpenCV library in Python

After this, the images were converted into numpy arrays and saved along with their labels.

Results

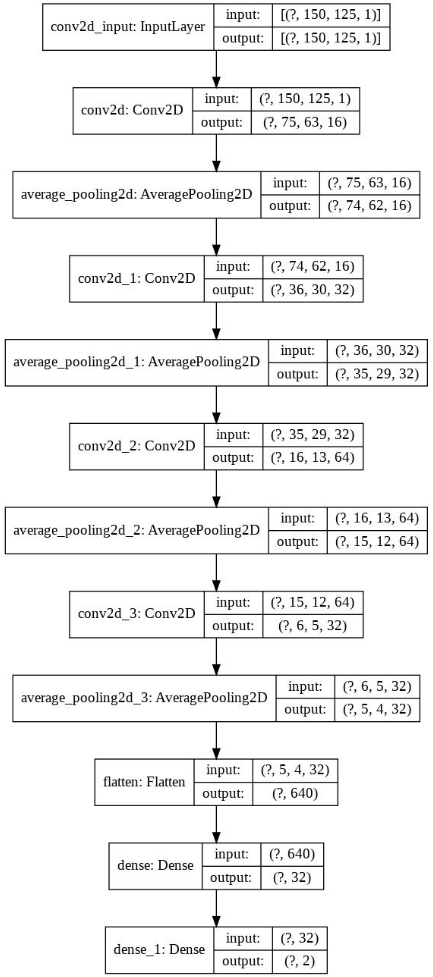
Model Description

We trained four models -

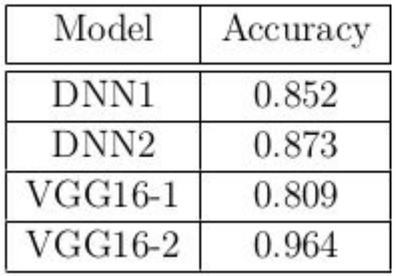
- A custom made DNN with 4 Conv2D layers and AveragePooling2D as the pooling layer (DNN1)
- The above architecture with MaxPooling2D replacing the Average pooling layer (DNN2)
- VGG16 architecture based model trained on the images after the preprocessing method 1 (VGG16-1)
- Same model but trained on images after preprocessing method 2 (VGG16-2)

The architecture of the custom DNN is shown in the next slide.

Model Architecture

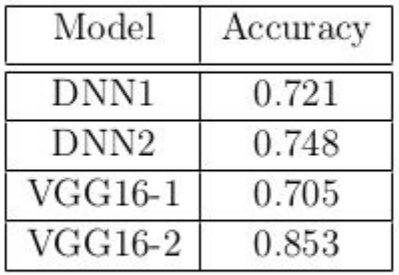

conv2d_input: InputLayer | input: [(?, 150, 125, 1)] | output: [(?, 150, 125, 1)]
conv2d: Conv2D | input: (?, 150, 125, 1) | output: (?, 75, 63, 16)
average_pooling2d: AveragePooling2D | input: (?, 75, 63, 16) | output: (?, 74, 62, 16)
conv2d_1: Conv2D | input: (?, 74, 62, 16) | output: (?, 36, 30, 32)
average_pooling2d_1: AveragePooling2D | input: (?, 36, 30, 32) | output: (?, 35, 29, 32)
conv2d_2: Conv2D | input: (?, 35, 29, 32) | output: (?, 16, 13, 64)
average_pooling2d_2: AveragePooling2D | input: (?, 16, 13, 64) | output: (?, 15, 12, 64)
conv2d_3: Conv2D | input: (?, 15, 12, 64) | output: (?, 6, 5, 32)
average_pooling2d_3: AveragePooling2D | input: (?, 6, 5, 32) | output: (?, 5, 4, 32)
flatten: Flatten | input: (?, 5, 4, 32) | output: (?, 640)
dense: Dense | input: (?, 640) | output: (?, 32)
dense_1: Dense | input: (?, 32) | output: (?, 2)

Model Evaluation on Test data



Model	Accuracy
DNN1	0.852
DNN2	0.873
VGG16-1	0.809
VGG16-2	0.964

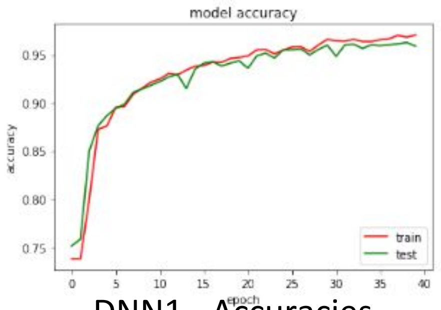
Test set results for 2-class classification



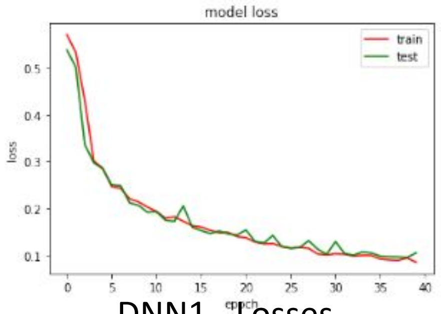
Model	Accuracy
DNN1	0.721
DNN2	0.748
VGG16-1	0.705
VGG16-2	0.853

Test set results for 2-class classification when training with Differential Privacy

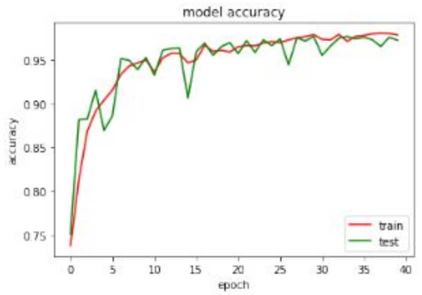
Model Performance



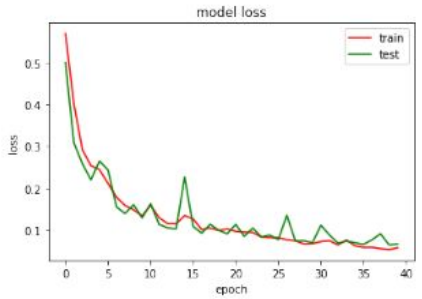
DNN1 - Accuracies



DNN1 - Losses



DNN2 - Accuracies



DNN2 - Losses

Model Evaluation

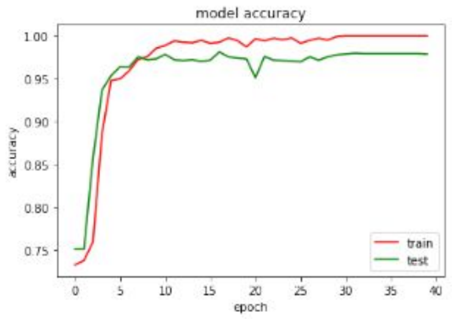


Figure 4.9: VGG16-1 - Accuracy

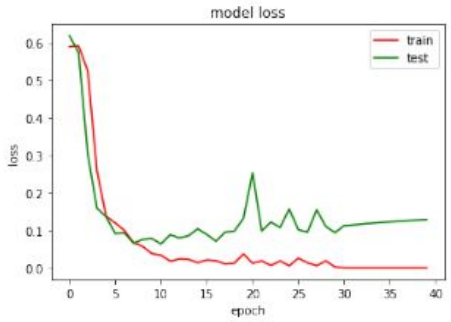


Figure 4.10: VGG16-1 - Loss

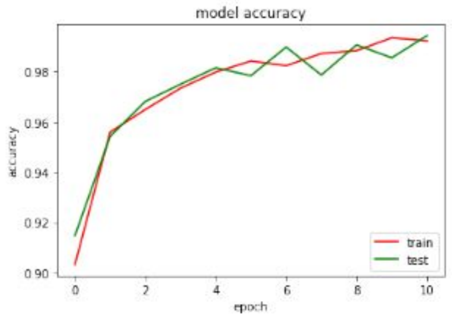


Figure 4.11: VGG16-2 - Accuracy

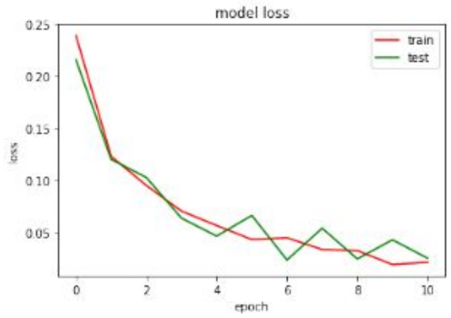


Figure 4.12: VGG16-2 - Loss

Results / Discussion

Model Evaluation on Test data

- The VGG16-based model when trained with the images generated using second preprocessing method is the best model in terms of both accuracy (96.41%) among all the other models.
- However, the VGG16 model trained on the data generated using first preprocessing method performs worse than the other two classes of models. It achieves an accuracy of 80.9%
- In case of the other two models, the one with MaxPooling (accuracy = 87.33%) performs better than the model with AveragePooling (accuracy = 85.2%).
- All these models were trained for 40 epochs with a ModelCheckpoint (in Keras) to save the best model in terms of validation accuracy and EarlyStopping (in Keras) to stop the training when the model performance tends to decrease.

Model Evaluation on Test data (contd.)

- Using Differential Privacy (DP) while training the models, results in a reduced accuracy (84.89%) which should also happen in theory as noise is introduced in the data while training.
- The accuracy (on an average) reduced by around 12% when training with DP.
- The DNN1 and DNN2 models were the most affected with reduction in accuracy of 13% where the VGG16 models saw an accuracy drop of 10% and 11% respectively.

Limitations and Future Work

Limitations

- TF Encrypted is a fairly recent framework developed in 2018.
- Some advanced Keras layers are still not implemented in that library or are buggy.
- Serving predictions from the encrypted models, is much slower than getting predictions from the unencrypted model. Although, this is expected, but for VGG16 it becomes very slow, which becomes very evident if having to be used as a service.

Future Work

- Work on the Github Issue created by us regarding the buggy and unimplemented layers.
- Crypten, a framework used by Facebook Research is also similar and very very new, hence also has a lot of bugs and buggy implementations.
- We can also experiment with more models provided some of the basic errors in the library are rectified.
- Furthermore, we can extend this to a general purpose medical image classification framework as well. Then it can be used to classify other such medical images as well.

THANK YOU!

Network Intrusion Detection in an Adversarial Setting

Aim - To fool Machine Learning based classifiers into falsely predicting malicious network traffic as benign

Intrusion Detection

- Dealing with unwanted access to systems and information by any type of user or hardware.
- Intrusion Detection System (IDS) is a device or software that monitors a network or systems for malicious activity or policy violations.
- There are two major categories of IDS -
    - **Network IDS -** These monitor network segments and analyze the network traffic to detect intruders.
    - **Host-based IDS** - These are installed in host machines and analyze processes, logs and other unexpected changes to detect malicious activity.

Adversarial Machine Learning

Adversarial Machine Learning (AML) is the study of machine learning in the presence of an adversary that works against the ML system in an effort to reduce its effectiveness or extract information from it. AML can be divided into two main types of attacks -

1. **Evasion (Exploratory) Attacks** - These attacks are performed on the **testing** phase. It does not tamper with ML models, but instead cause it to produce adversary selected outputs
2. **Poisoning (Causative) Attacks -** These attacks are performed on the **training** phase. Attackers attempt to learn, influence, or corrupt the ML model itself.

Adversarial Example Generation

Research has been done on the methods and algorithms to generate adversarial examples. There are many such methods which have a trade-off on speed of production, performance and complexity. Some of the methods that have been proposed are given below -

- **Evolutionary algorithms** - Proposed by Nguyen et. al. in 2015. But this method is very slow compared to the other two alternatives described below.
- **Fast Gradient Sign Method (FGSM)** - Proposed by Goodfellow et. al. in 2014
- **Jacobian-based Saliency Map Attack (JSMA) -** Proposed by Papernot et al. in 2016. This method is more computationally expensive than FGSM but it has the ability to create adversarial samples with less degree of distortion.

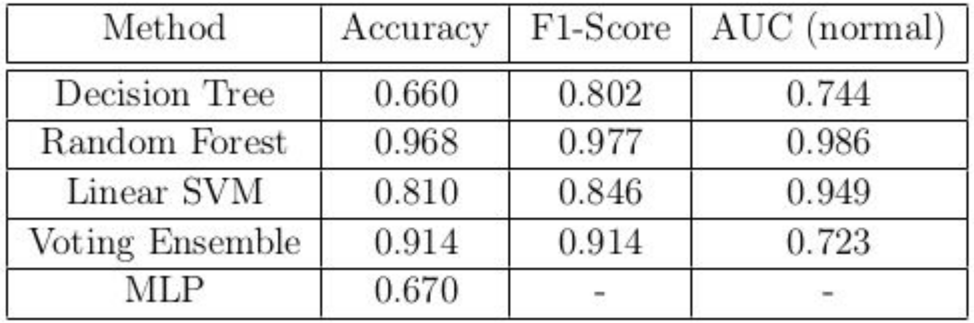
Dataset - NSL KDD

The attacks that are present in the datasets can be divided into four major categories -

- **Denial of Service (DoS) Attacks** attacks are an interruption in an authorized user’s access to a computer network, in other words, they are attacks against availability. This category contains attacks such as *smurf*, *neptune*, *mailbomb*, *udpstorm*, etc.
- **User to Root (U2R) Attacks** indicate attempts of privilege escalation. Some attacks of this type in the dataset are *buffer overflow, loadmodule, sqlattack* and *rootkit*.
- **Root to Local (R2L) Attacks** attacks aim to gain remote access to a system by exploiting a vulnerability. Some examples of this type of attacks are *multihop, guesspasswd, httptunnel and xsnoop.*
- **Probe** attacks aim to gather information by using enumeration techniques like scanning or probing different parts of the network, for e.g. the ports. Some examples of such types of attacks are *ipsweep, portsweep, nmap and mscan.*

Model Evaluation on Adversarial data

- This section presents the results of the baseline models on the adversarial test set generated by the JSMA method in terms of Accuracy, F1-score and AUC for the ROC curve.
- One thing to note here is that, both the AUC results as well as the ROC curves in the figures below, are only presented for the the "normal" class, while the F1-score is an average score over all classes.



Method	Accuracy	F1-Score	AUC (normal)
Decision Tree	0.660	0.802	0.744
Random Forest	0.968	0.977	0.986
Linear SVM	0.810	0.846	0.949
Voting Ensemble	0.914	0.914	0.723
MLP	0.670	-	-