Predicting Precipitation Patterns: Deep Learning for Rainfall Prediction

Aabha Rudrabhate, Srinath Divate, Tapasvi Taktode, Anurag Tekam, Yatharth Thakare

***Abstract – asasdafafaasfanfkdsbjsdbkdbskvjbdsjobfoabfklsd vkndsbjb fbsb ksd vjsd bvidbsfj jfndAOfbsdvojsDBvjsdbo aof aDvns’dvn***

I. INTRODUCTION

Deep Neural Networks (DNN) have been effectively used in a wide range of fields, including medical imaging, video analysis, language modelling and translation, picture classification, and meteorology. The use of DNNs to the unsupervised representation issue of weather forecasting has gained significant attention in recent times. Next-frame prediction, which predicts potential future pictures by providing historical image information, is a novel and promising avenue of computer vision research in these sorts of situations. Weather forecasting refers to the state of art of predicting future conditions such as precipitation, temperature and wind. We are specifically interested in the field of "nowcasting," which is the name for high resolution, short-term (i.e., 0 to 2 hours) precipitation or other meteorological quantity weather forecasts. The precipitation nowcasting field helps in the accurate prediction of rainfall over an area by looking at radar images. The field deals with the generation of the radar image at some points in the near future.

One of the various architectures researchers have tried to implement is Conv-LSTM [1] that has shown great results in dealing with time-series data due to them being pretty good at extracting patterns in input feature space, where the input data spans over long sequences. Because LSTMs have a gated design that allows them to control their memory state, they are perfect for solving these kinds of issues. With the continuous input of data from one end, we know that the prediction needs to be swift while dealing with a huge amount of data. With the same thing in mind, some recent research introduced UNet to counter the same issue and were successful to get good results on image-to-image translation problems [3, 4].

Usage of conditional GANs (cGANs) [6] has became widely popular under the domain of image-to-image translation. As we require a model that would learn from previous inputs in the sequence, we tried to make use of this property where cGANs predicts N future radar frames given M past-conditional frames. In this methodology, we also introduced encode-decoder architecture with the Axial Transformer in our Novel model, a simple yet effective self-attention-based autoregressive model for data organized as multidimensional tensors.

"Axial attention" is the term used to describe how Axial Transformer applies attention along a single tensor axis without flattening, as opposed to a flattened string of tensor elements. An axial attention operation benefits from a large reduction in computation and memory compared to ordinary self-attention since the length of any single axis, or the height or breadth of an image, is usually significantly lower than the total number of components. By using the Analytical Transformer framework to train the encoder-decoder, we are able to get cutting-edge outcomes. Our contributions are summarized as

follows:

* We implemented an encoder decoder architecture and passed the frames of videos as input to the architecture.
* The output of encoder decoder architecture is passed as input to the Axial Transformer network.
* With sequence frames given as input, the final output of the UNet with Axial Transformer model is a distribution, out of which we select the mean value for every pixel.

II. RELATED WORK

Previously, the Conv-LSTM model has shown promising results on the next frame prediction problem on the Moving MNIST dataset [1]. The dataset was initially created in the context of Unsupervised Learning of Video Representations [2] and used LSTM to learn the representation of video sequence. As the radar images too are a kind of time series data and we know that clouds can not abruptly change direction or disappear, we know that there are some motion parameters associated with a continuous sequence of radar images too. Using the same idea of next frame prediction in a video sequence, ConvLSTM can be applied to a dataset of radar images too. UNet is one of the recent models that has showcased its ability to predict the next frame of time-series data very well [3]. In the paper, as part of Traffic4cast challenge 2019, UNet was used to predict short-term traffic flow volume. The input and output of the model were the same sizes that are also relevant in our context as we need to reproduce radar images of the same location sometime in the future. Another paper that actually implemented UNet for the weather forecasting problem [4] saw a significant increase in results on the dataset consisting of precipitation maps from a region of the Netherlands and a binary image of cloud coverage of France. The size of the model here is very small compared to previous models that were used to solve the same problem, which is also a significant advantage considering the latency requirement of our problem statement.

III. STUDY INTRODUCTION

For this study, two different approaches of machine learning were compared to evaluate how different models can predict potential malware files. The study was performed in different phases. The first phase consisted of collecting real-world data. The second phase consisted of preprocessing the dataset to make it suitable for training. The third phase consisted in developing simulation software where the models can be tested with different settings and approaches.

IV. METHODS

*A. Data Collection*

The data used for this research paper is structured in a tabular way. The dataset contains important features extracted from both malicious and benign files. The collection process began with malicious files. Data was collected from datasets provided by previous research [4]. A total of 3800 benign files were parsed, most of them obtained from Win32 as well as already known benign third-party files (e.g., desktop applications).

The files used for this study use the Windows Portable Executable (PE) format (shown in Fig. 1). This format contains sets of information or modules that are used by the operating system for execution. Hackers target these headers so they can store and spread malicious content [5]. As a result, the different features found in the headers are essential to develop a machine learning model that can identify this kind of software. In fact, most of the previous research on the most important features found in PE files suggest including almost all the characteristics found in the headers. For this study, we included the PE Header features as well as the following sections: data, text, pdata, rdata, rsrc, and reloc.

|  |
| --- |
| DOS HEADER |
| PE HEADER |
| OPTIONAL  HEADER |
| SECTION TABLE |
| SECTIONS  (e.g., code, data) |

**Fig 1**. Different parts of a Windows Portable Executable

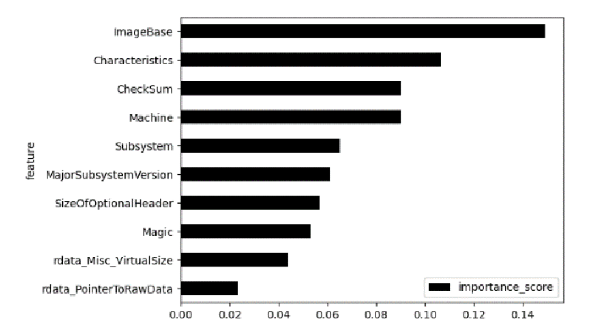
All sections (except section table) were used for this study.

*B. Data preprocessing*

It is worth mentioning that the malware dataset had as a label the type of malware the file was. Since this problem’s objective is to find out whether the file is malware or not, the dataset was modified to have a binary categorical feature representing the type of software (i.e., Benign or Malicious).

Additional preprocessing techniques were applied. To normalize continuous features, a range normalization from 0 to 1 was applied. To reduce noise of data, IQR 1.5 was used to clamp outliers.

Overall, the dataset had more than 100 features. However, having many features makes it more possible to have an overfitting model as a result [6]. Overfitting occurs when the prediction model “memorizes” the training set. As a result, while the prediction model works with test data from the original dataset, it is harder for it to work with new data such as real-world applications. To reduce this, feature selection was applied to the dataset. This was performed by using a Random Forest Classifier to create a prediction model and consecutively identify the features’ importance score. The dataset features amount was reduced from 106 to 10 features (Shown in Fig. 2).

**Fig 2.** Top 10 most important features in the dataset.

*C. Models Settings*

One of the purposes of the study was to identify which machine learning model was the most accurate and efficient to be deployed. It is important to mention that some models were used in only one approach (i.e., centralized or federated). This is because the model’s nature might not always be compatible with a federated approach. The study will explore the following models:

1. Gaussian Naïve Bayes: Probabilistic approach that assumes the distribution of data is gaussian (normal). Used in Centralized approach. No special parameters used.

2. K-Nearest-Neighbors: Similarity based approach that classifies based on proximity. Used in Centralized approach. For results collection, the model was performed with 4 neighbors and using Manhattan distance.

3. Logistic Regression: Error-based approach derived from Linear Regression. Used for classification tasks. Used in both Centralized and Federated. For results collection, the model was performed with max iterations of 100 for centralized approach and 10 for federated learning. Additionally, a learning rate of 0.002 was used.

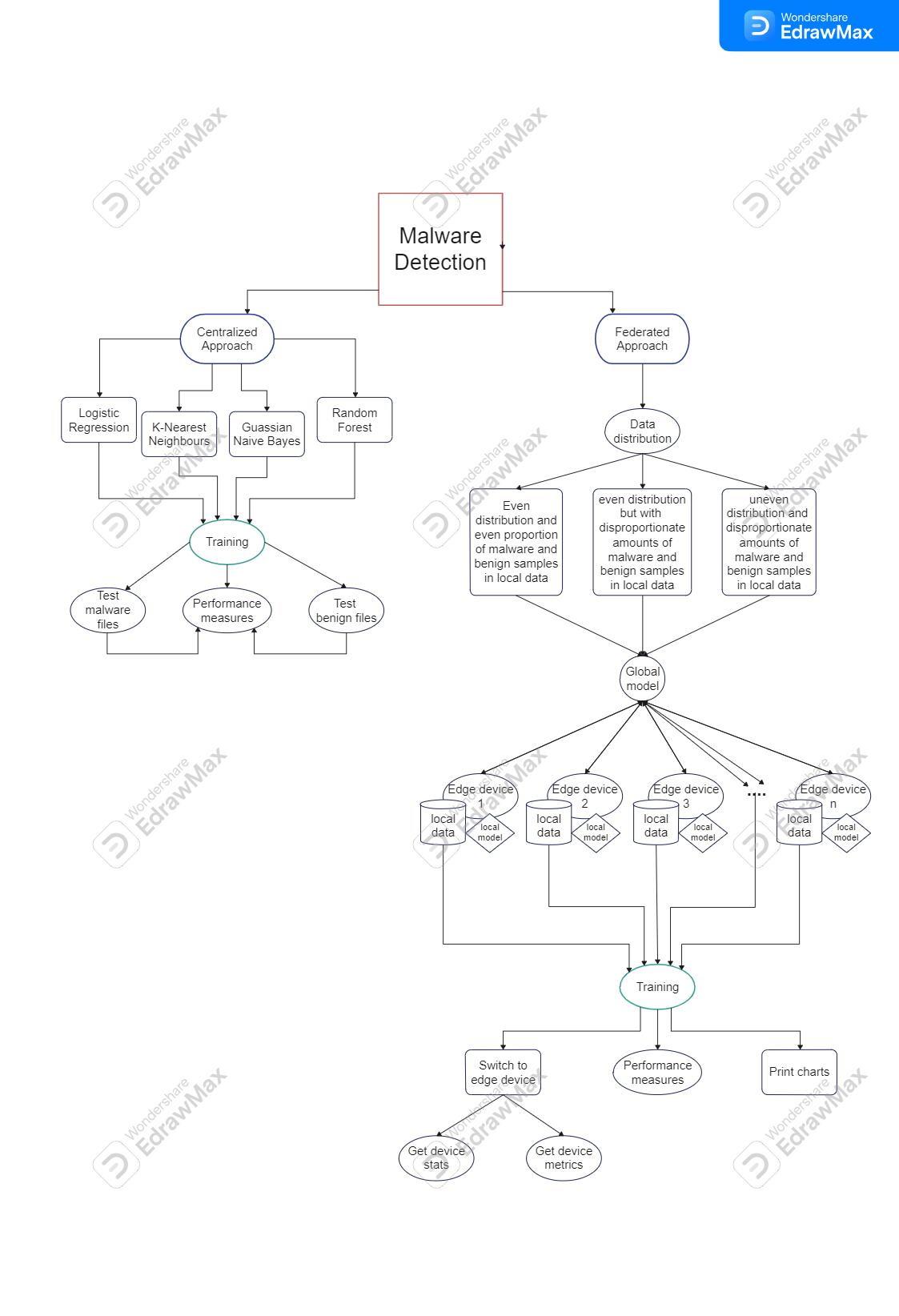
4. Random Forest: Learning based approach. Ensemble of Decision Trees. Used in Centralized approach. Parameters used in the study consisted of the number of estimators being 100, max depth being 5, and using Gini as criterion for importance score.

*D. Federated Approach explanation in detail*

The traditional way of using machine learning involves storing the data and models in a centralized server. However, this approach has some disadvantages. The high communication and storage costs, along with data privacy, will increasingly challenge the traditional eco-system of centralized over-the-cloud learning and processing for IoT platforms [7]. On the other hand, the federated learning approach aggregates only the models, not the data. This brings potential benefits such as preserving the privacy of user data, improving model performance, and having a more flexible scalability [7].

The federation process can be divided into 4 steps: A subset of contributing models is randomly selected from all the edge devices. Second, the global model is deployed from the server to the devices. Third, the devices train their models with their local data. Fourth, the devices send their trained models for aggregation in the central server. This process is repeated every iteration (also called federated round).

There are different approaches for model aggregation. One of the most popular techniques is called Federated Averaging. This approach consists of averaging the different local model parameters to set into the new global model every iteration [8]. For Logistic Regression, the chosen model for the study of this approach, aggregation consisted of weighted averaging of the models’ coefficients and bias. The model’s weight was determined by their local dataset size.



**Fig 3**.System flowchart

*E. Simulation Software*

The software was developed in Python. Users can interact with the program through the Command Line Interface. To use machine learning algorithms some libraries were needed (i.e., sklearn, NumPy, and pandas). Simulation takes two steps: Training and Testing.

On the training stage, the user can choose their desired machine learning model (e.g., Naïve Bayes) as well as their preferred approach (traditional or federated). Then, using sklearn, datasets are partitioned into training and test set. Sklearn then trains the machine learning model and shows the current model accuracy.

The testing phase consists of the user testing the performance of the machine learning model. They can select from preselected known malware examples. The simulation also lets the user see the most optimal settings of their chosen machine learning model according to the previously set parameters.

The software also includes features to visualize the collection of various types of collected data (e.g., Visualizing metric improvement overtime in Federated Learning).

*F. Simulation Settings*

Since the data is also centralized on the Centralized approach, it was assumed that the amounts of Benign and Malicious samples were equally distributed. The number of used samples was 6000 (3000 malware samples and 3000 benign samples). The partition of training and test datasets always was 75% of the dataset and 25% of the dataset respectively.

As mentioned previously, Federated Learning is characterized by decentralization of data. As a result, it is fair to assume that most likely there will be edge devices with imbalanced data. To account for this, we decided to explore how the approach works in three different cases:

• Equal distribution (i.e., all edges have the same amount of data as well as the same proportion of malware/benign samples).

• Balanced distribution with imbalanced local malware/benign proportion

• Imbalanced distribution along with imbalanced local malware/benign samples

The specific settings used to generate the results for the three cases were number of edge devices = 20, learning rate=0.02, and number of iterations = 750. However, different executions of the software might give different results as the amount each device has is assigned randomly.

For this approach, we used 7% of the whole dataset as a testing dataset and 93% as the training set to be split among the edge devices. The metrics were calculated based on the performance of the same testing dataset.

*G. Evaluation Metrics*

The metrics used to evaluate the models (both centralized and federated) are the following:

• Accuracy: Measures how many predictions were correct.

• Recall: Measures how confident we can be that all the instances with the positive target level have been found by the model [9]. In the problem context, this means the proportion of actual malware correctly identified by the model.

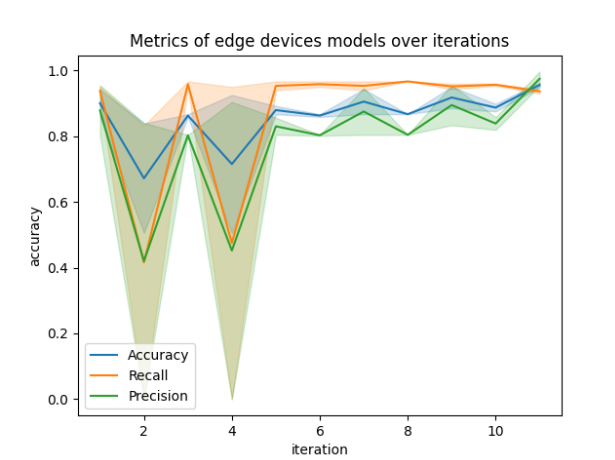
• Precision: Measures how confident we can be that an instance predicted to have the positive target level has the positive target level [9]. In the problem context, this means the proportion of predicted malware that is malicious. This is an important metric to consider, as it can tell how prone the model is to predict false positives, which can be an inconvenience for the user and the system.

V. RESULTS

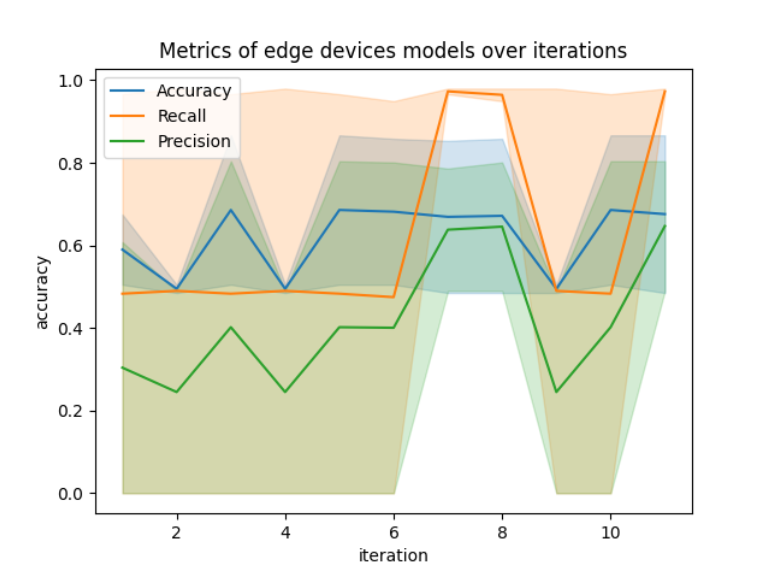
TABLE I. CENTRALIZED MODELS PERFORMANCE

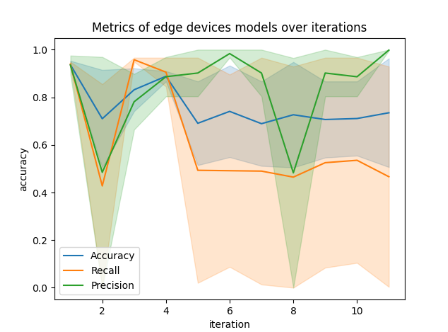
|  |  |  |  |
| --- | --- | --- | --- |
| Model/Metric | Accuracy | Recall | Precision |
| Gaussian  Naïve Bayes | 91.8% | 97.4% | 87.8% |
| K Nearest  Neighbors | 98.9% | 98.7% | 99.1% |
| Logistic  Regression | 95.3% | 95.5% | 95.1% |
| Random  Forest | 98.9% | 99.3% | 98.4% |

According to Table I, the best performing models were Random Forest and K-Nearest Neighbors. While their accuracy performance was the same, Random Forest had higher recall and K-Nearest Neighbors had a higher precision.

The other objective of the study was to explore how the federated learning approach could perform in terms of malware detection with machine learning. To generate these results, Logistic Regression was the only model used with this approach. 

**Fig 4.** Metrics of edge devices with an even distribution and even proportion of malware and benign samples among local data.

**Fig 5.** Metrics of edge devices with an even distribution but with disproportionate amounts of malware and benign samples among local data.

**Fig 6.** Metrics of edge devices with uneven distribution and disproportionate amounts of malware and benign samples among data.

VI. DISCUSSION

Regarding the centralized approach, we can appreciate that the dataset works best with Information-based and Similarity-based machine learning models. Files of a certain nature might have features similar to the ones found on malware.

The results on the federated logistic regression show that Federated Learning is a potentially good approach for malware detection. Comparing the different figures (4, 5, 6), we can see a pattern. As the distribution of the data gets more diverse among the edge devices, the iterations taken for the models to converge increase. Despite this, we could appreciate how the federated approach progressively improved the devices metrics and eventually stabilized them. This is an important observation as in a real-world scenario data is more likely to be unevenly distributed among devices as well as unevenly proportioned in terms of malware and benign samples.

VII. CONCLUSION

Based on the previous findings, we can conclude that both the centralized and the federated approaches have their advantages and limitations when it comes to malware detection. The centralized approach works best with Information and Similarity based learning models such as Random Forest and K-Nearest Neighbors. However, there are things yet to improve, such as adding many more samples of files of different nature (e.g., game-related files) to reduce misclassification.

The federated learning approach showed promising results for malware detection, indicating it can be a potentially good approach in the future. The study revealed a pattern in which the diversity of distribution and proportion of data increased the number of iterations required for convergence. Despite this, the approach gradually improved the devices metrics and stabilized them for all three presented cases, showing that the approach might be suitable for real-world application.

VIII.FUTURE WORKS

As mentioned before, both the centralized and federated approaches showed positive metrics in malware detection. However, there is still room for improvement. Some of the extensions of the research presented can include:

• Using the models on files from a greater variety of nature such as entertainment apps, video games, and other high-performance programs to visualize better which areas are the most misclassified.

• Using larger amounts of samples of these natures to see whether misclassification is reduced.

• Taking an input file from a user in real-time and classifying it as malware or benign.

• Exploring more models with different approaches such as Neural Networks.

• Using the models on known new malware files to see how effective the approach is beyond the dataset.

REFERENCES

[1] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional lstm network: A machinelearning approach for precipitation nowcasting, 2015.

[2] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms, 2016.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[4] Kevin Trebing, Tomasz Stanczyk, and Siamak Mehrkanoon. Smaat-unet: Precipitation nowcasting using a small attention-unet architecture, 2021.

[5] Suman Ravuri, Karel Lenc, Matthew Willson, Dmitry Kangin, Remi Lam, Piotr Mirowski, Megan Fitzsimons, Maria Athanassiadou, Sheleem Kashem, Sam Madge, and et al. Skilful precipitation nowcasting using deep generative models of radar. Nature, 597(7878):672–677, Sep 2021.

[6] Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling, 2021.

[7] Sungbin Choi. Traffic map prediction using unet based deep convolutional neural network, 2019.

[8] Casper Kaae Sønderby, Lasse Espeholt, Jonathan Heek, Mostafa Dehghani, Avital Oliver, Tim Salimans, Shreya Agrawal, Jason Hickey, and Nal Kalchbrenner. Metnet: A neural weather model for precipitation forecasting, 2020.