



Project Report

On

YouTube Tab Classification and User Privacy using Federated Learning

Submitted in partial fulfillment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

Ankita Rajput

2020BITE030

Ishita Sharma

2020BITE013

Under the supervision of

Dr. Prabal Verma

Department of Information Technology
National Institute of Technology Srinagar

July 2024



CERTIFICATE

This is to certify that the project titled **YouTube Tab Classification and User Privacy using Federated Learning** ‘ has been completed by **Ankita Rajput (2020BITE030)** and **Ishita Sharma (2020BITE013)** under my supervision in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Information Technology**. It is also certified that the project has not been submitted or produced for the award of any other degree.

Dr. Prabal Verma

Supervisor

Dept. of Information Technology

NIT Srinagar



STUDENTS' DECLARATION

We, hereby declare that the work, which is being presented in the project entitled **YouTube Tab Classification and User Privacy using Federated Learning** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Information Technology**, in the session 2024, is an authentic record of our own work carried out under the supervision of **Dr. Prabal Verma**, Department of Information Technology, National Institute of Technology, Srinagar. The matter embodied in this project has not been submitted by us for the award of any other degree.

Dated: _____

(i) Name: Ankita Rajput

Signature: _____

(ii) Name: Ishita Sharma

Signature: _____

ACKNOWLEDGEMENT

It is our privilege to express our sincerest regards to our department head **Dr. Janibul Bashir** for encouraging and allowing us to present the project on the topic **Youtube Tab Classification and User Privacy using Federated Learning** at our department premises for the partial fulfillment of the requirements leading to the award of B.Tech degree.

We deeply express our sincere thanks to our project guide, **Dr. Prabal Verma**, for his valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of this project.

We take this opportunity to express gratitude to all of the department faculty members for their help and support.

We also place on record, our sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture.

Ankita Rajput (2020BITE030)

Ishita Sharma (2020BITE013)

Abstract

E-learning, which employs electronic technologies like computers, mobile devices, and the internet, has seen a significant rise in popularity recently. While it offers the potential to provide education globally, it also poses risks for time and resource wastage. Students frequently use the same devices for both studying and entertainment, making it challenging to resist the lure of social media and other distractions. This issue is particularly relevant as online education becomes more prevalent especially after Covid-19. Surprisingly, there has been little research into monitoring students' screen activities while also protecting their privacy. Therefore, our study introduces a privacy-preserving approach using federated learning to determine whether students are spending their time productively on their devices or getting distracted, all while ensuring their privacy is maintained.

Contents

List of Tables	vii
List of Figures	viii
Abbreviations	ix
1 Introduction	1
1.1 Motivation	2
1.2 Problem Statement & Objectives	2
1.3 Organization	3
2 Literature Survey	4
3 CNN and Federated Learning: A brief Introduction	8
3.1 Overview of Federated Learning	8
3.1.1 Privacy Preservation with Federated Learning	9
3.2 Overview of CNN	10
3.3 Types of CNN	10
3.3.1 InceptionV3	10
3.3.2 FedCNN	11
3.3.3 InceptionResNetV2	11
3.4 InceptionV3 in Classification	11
4 Implementation	12
4.1 Introduction	12
4.2 YouTube Tab Classification and User Privacy using Federated Learning	12

4.2.1	Dataset	12
4.3	Model Architecture	13
4.3.1	Model Comparison	14
4.3.2	Train Global and Local Model	14
4.3.3	Perform Detection	15
4.3.4	Creating Local Dataset	15
4.3.5	Update Global Model	16
4.4	Conclusion	17
5	Experiment and Result	18
5.1	Experiment Setup	18
5.2	Model Evaluation	18
5.2.1	Conclusion	21
5.3	Result	22
5.3.1	Classification	22
5.3.2	Folder Creation	23
5.3.3	Update Global Model	24
6	Conclusion & Future Work	25
6.1	Conclusion	25
6.2	Future Work	26
	Bibliography	27
	Appendices	29

List of Tables

2.1	Literature Review	7
3.1	Comparison between Privacy Preserving Techniques	9
4.1	Dataset Distribution	13
5.1	Model's performance metrics	21

List of Figures

4.1	Workflow of Model	13
4.2	YouTube Tab Watcher	15
4.3	YouTube Tab Detector	16
5.1	Accuracy	19
5.2	Precision	19
5.3	Recall	20
5.4	Area under curve	20
5.5	Precision-Recall Curve	21
5.6	Take screenshot and classify it	22
5.7	Classification of screenshot	22
5.8	Screenshot folder	23
5.9	Screenshot classified	23
5.10	Screenshot saved	23
5.11	Update Global Model	24

Abbreviations

CNN	Convolutional Neural Network
FD	Federated Learning
AUC	Area under Curve
PRC	Precision Recall Curve
API	Application Programming Interface

Chapter 1

Introduction

As the fourth industrial revolution progresses, the internet is becoming faster and more accessible, making digital devices crucial for both education and professional work. Online education has grown exponentially, overcoming geographical barriers and offering flexible learning environments. In 2016, over 6 million US students enrolled in at least one online course, highlighting the increasing demand for accessible education. The global e-learning market is projected to reach \$325 billion by 2025 [2].

However, maintaining student engagement in online learning presents unique challenges. One significant issue is the potential for time and resource wastage, especially since students often use the same device for both studying and entertainment. Social media and other distractions, particularly YouTube, are just a click away. Unintentionally, students may spend valuable time on YouTube watching non-educational content instead of focusing on their studies. Research by S. Khan et al [9]. and Geot [4] shows that excessive social media use negatively impacts academic performance. A study involving 379 students from universities in the Khyber Pakhtunkhwa region found that most students perceived social media, including YouTube, to adversely affect their academic performance [8].

The amount of time spent on these platforms correlated with negative perceptions of their impact on academics. Traditional social media blockers often fail as they rely on self-discipline and can be bypassed. To address this, our research proposes a privacy-preserving system using federated learning to specifically monitor YouTube usage. By analyzing screen content, we aim to distinguish between educational and non-educational activities, ensuring students utilize their time effectively

without compromising their privacy. Our approach keeps data local and leverages transfer learning to adapt models efficiently, aiming to develop a robust, secure, and privacy-centric system for online learning environments.

1.1 Motivation

In today's digital age, understanding user behavior on platforms like YouTube is crucial for improving user experience and personalization. However, traditional methods of data analysis often involve collecting and centralizing vast amounts of user data, which raises significant privacy concerns. According to previous research, the main aim was to provide detailed insights into user behavior and interactions with their devices.

This project is crucial because it not only improves the accuracy of the model but also upholds user privacy, which is increasingly important in today's data driven environment.

1.2 Problem Statement & Objectives

Our problem statement was to design a system such that it classifies the YouTube tab of client while maintaining the user-privacy using federated learning.

The objectives at the same time, were to ensure the following:

1. Distinguish between productive and unproductive activities on YouTube to ensure effective time utilization.
2. Develop a privacy-preserving system using federated learning to preserve the privacy of end user.
3. Update global model by using simple voting from the weights achieved from various local models.
4. Robust, secure, and privacy-centric youtube activity detection system for online learning environments.

1.3 Organization

The rest of the report is organized as follows. We review the existing literature in Chapter 2. In chapter 3, we introduce the concepts behind **Federated Learning** and **CNN**. In chapter 4, we have discussed the implementation details of the project, explaining all its features, and specified the code snippets wherever required. In chapter 5, we have shown the results and experimentation of our project. Finally in chapter 6, we have briefly summarized the project, and given directions for future work.

Chapter 2

Literature Survey

In this chapter, we review recent literature to understand the current research in the field. By analyzing each paper, we made key observations and inferences, which laid the foundation for our project.

Computer screen activity detection has gained considerable attention in recent years, particularly for user behavior analysis and remote learning. Advances in computer vision, machine learning, and deep learning have significantly improved activity detection. However, privacy-preserving techniques, especially for YouTube activity detection, remain underexplored. This literature review discusses state-of-the-art approaches in privacy-preserving methods and YouTube activity detection.

Using screen capture to study user research behavior: [3] The paper by B. Imler and M. Eichelberger details a method using video screen capture technology to study user research behavior in library databases. The model works by capturing real-time video of users' computer screens as they interact with library resources. This non-intrusive method allows researchers to observe and analyze user behaviors and patterns without physical presence, thus avoiding the influence that direct observation might have. The captured videos can be easily exported into coding software for detailed data analysis, making it a practical and efficient tool for usability studies.

The researchers employed screen capture software to monitor and record the on-screen actions of users as they conducted research. This approach allowed for an unobtrusive observation of user behavior in a naturalistic setting. Screen captures provided a rich dataset that included not only

the websites and databases users visited but also their search queries, navigation patterns, and the duration spent on different tasks. This level of detail offered insights that traditional survey or interview methods might miss.

The findings from this research have practical implications for improving library services and digital interfaces. By understanding user behavior in detail, libraries can design more user-friendly systems and provide better support for research activities. The study suggests that libraries could use similar screen capture techniques to continually assess and refine their digital services.

Track every move of your students: Log files for Learning Analytics from mobile screen recordings: [5] The paper by P. Krieter and A. Breiter, titled “Track Every Move of Your Students: Log Files for Learning Analytics from Mobile Screen Recordings,” explores the use of log files generated from mobile screen recordings to enhance learning analytics. By capturing and analyzing these recordings, the study aims to gain insights into students’ digital interactions and learning behaviors. The authors discuss how this data can help improve educational strategies and personalize learning experiences.

Krieter and Breiter utilized mobile screen recording technology to capture the interactions of students with educational applications and resources. This approach provided a detailed, real-time account of student activities on their mobile devices. The screen recordings were converted into log files, which systematically documented every action taken by the students, including taps, swipes, and navigation through the app. These logs served as a comprehensive dataset for analysis.

By examining the detailed logs, educators and researchers could better understand the learning processes and difficulties faced by students. The data helped in identifying common challenges, misconceptions, and effective strategies used by students, thus informing instructional design and personalized learning interventions.

The study also addressed the ethical considerations related to privacy and consent. Ensuring that students’ personal information was protected and that they consented to the recording of their activities was emphasized as crucial for ethical research practices.

The insights gained from this approach have the potential to significantly enhance learning analytics. Educators can use the data to tailor educational content to better suit individual student needs, develop targeted interventions, and improve overall learning experiences.

Tracking digital device utilization from screenshot analysis using deep learning: [1] The paper by B. J. Ferdosi, M. Sadi, N. Hasan, and M. A. Rahman introduces a novel methodology to monitor and analyze digital device usage through the examination of screenshots. The study utilizes deep learning techniques to process and interpret the visual content captured in screenshots, aiming to uncover patterns and behaviors in digital device interaction. The proposed approach involves the extraction of relevant features from screenshots and the application of convolutional neural networks (CNNs) to classify and analyze the data. This technique enables the automatic identification of applications in use, user activities, and time spent on various tasks. The paper details the dataset preparation, model training, and evaluation processes, demonstrating the effectiveness of deep learning in providing detailed insights into device utilization. This method offers significant implications for understanding user behavior, enhancing user experience, and developing more efficient digital interfaces.

The primary goal of this research paper was to develop a method for tracking and analyzing digital device usage by leveraging deep learning models to process and interpret screenshots captured from these devices. The researchers employed deep learning algorithms, particularly convolutional neural networks (CNNs), to analyze the content of screenshots. These models were trained to recognize various elements and activities depicted in the screenshots, such as app usage, browser activity, and other on-screen interactions.

Screenshots were collected from a variety of digital devices, encompassing different usage scenarios and contexts. This dataset was then annotated to identify key features and activities present in the images, providing a labeled dataset for training the deep learning models. The model was trained to classify and identify different types of device utilization activities, extracting meaningful patterns from the visual data.

The deep learning-based analysis enabled the extraction of detailed information about user behavior and device interactions. Insights included the frequency of app usage, time spent on different activities, and patterns of multitasking. The study also addressed the importance of protecting user privacy when collecting and analyzing screenshots. Ensuring that personal information is anonymized and that users consent to data collection was highlighted as essential for ethical research practices.

Table 2.1: Literature Review

Paper	Video Recording	Screenshot	Dataset	Privacy Preservation
Using screen capture to study user research behavior [3]	✓	✗	✗	✗
Track every move of your students: Log files for Learning Analytics from mobile screen recordings [5]	✓	✗	✓	✗
Tracking digital device utilization from screenshot analysis using deep learning [1]	✗	✓	✓	✗
A survey on federated learning: challenges and applications [7]	✗	✓	✗	✓
A comprehensive survey on federated learning concept and applications [6]	✗	✓	✗	✓

Chapter 3

CNN and Federated Learning: A brief Introduction

3.1 Overview of Federated Learning

Federated Learning is a decentralized machine learning approach where multiple clients (e.g., devices or servers) collaboratively train a model while keeping the data localized. This method enhances privacy and security by ensuring that raw data never leaves the local devices. Instead, only model updates (e.g., gradients) are shared with a central server, which aggregates these updates to improve the global model.

This process preserves privacy, reduces communication overhead, and allows for personalized local models. FL is particularly beneficial in scenarios where data is inherently decentralized, such as on smartphones or IoT devices. Key applications include healthcare, finance, IoT, and telecommunications, where sensitive data can be utilized for model training without compromising privacy. Despite its advantages, FL faces challenges like communication costs, data heterogeneity, system heterogeneity, security concerns, and scalability issues. Overcoming these challenges is essential for the widespread adoption of FL, which holds significant potential for advancing privacy-preserving collaborative learning across various industries.

3.1.1 Privacy Preservation with Federated Learning

Federated Learning protects user privacy by keeping data on individual devices rather than uploading it to a central server. In traditional machine learning, screenshots would need to be sent to a central cloud for training, which risks security breaches and exposes user privacy. FL avoids this by allowing models to be trained on decentralized data sources without centralizing the data. The data stays on the devices or servers that generate it, and the model is trained locally. Only the updated model weights are shared between the devices and the central server, not the data itself. This approach ensures that individual data sources remain private and secure, enabling effective model training without compromising data privacy.

In this decentralized model training architecture, data is initially read at the input layer. To ensure compatibility, all images are resized to a fixed shape with specific pixel values, as the current architecture supports only identical data. The dataset is then randomly distributed among clients. A global training loop is established where each client receives a copy of the global model, and the local model weights are set to match those of the global model. The local model is trained, tested, and validated using local data. Subsequently, the local weights are collected, averaged, and the new weights are set for the global model, which is then sent back to the clients.

Privacy Preserving Techniques	Accuracy	Utility	Privacy	Computational Complexity
Federated Learning	High	Medium	High	Low
Differential Privacy	Low	Low	High	Low
Homomorphic Encryption	High	High	Medium	High

Table 3.1: Comparison between Privacy Preserving Techniques

3.2 Overview of CNN

A Convolutional Neural Network is a specialized type of deep learning model designed primarily for analyzing visual data such as images or video frames. It operates by systematically applying learnable filters across small regions of the input data, known as convolutional layers. These filters enable the network to automatically extract hierarchical representations of features like edges, textures, and patterns. Following convolutional layers, pooling layers downsample the feature maps to reduce computational complexity and increase the network's ability to generalize to variations in input data.

Non-linear activation functions, such as ReLU, introduce non-linearity into the model, allowing it to learn complex relationships. In CNNs, fully connected layers integrate these extracted features to make final predictions, connecting every neuron in one layer to every neuron in the next. Throughout training, CNNs use backpropagation and gradient descent to adjust model parameters, optimizing them to minimize prediction errors. CNNs are widely applied in tasks such as image classification, object detection, facial recognition, medical image analysis, and natural language processing, demonstrating their effectiveness in extracting meaningful information from visual data.

3.3 Types of CNN

3.3.1 InceptionV3

Inception v3 is a CNN architecture known for its efficient and accurate performance in image processing tasks. It introduces Inception modules that utilize parallel convolutional layers of different sizes and pooling operations to capture diverse features at multiple scales within each layer. The network incorporates factorization techniques to optimize computational efficiency, along with batch normalization and ReLU activation for improved training speed and convergence. Widely used for image classification, object detection, and scene recognition, Inception v3 has demonstrated superior performance in various computer vision applications due to its effective utilization of multi-scale feature extraction and architectural optimizations.

3.3.2 FedCNN

FedCNN, or Federated Convolutional Neural Network, adapts CNN architectures for federated learning setups where multiple decentralized clients train local models on their data without sharing it centrally. Each client's local model updates are aggregated periodically to refine a global model, ensuring collaborative learning while preserving data privacy. FedCNN optimizes communication by transmitting compressed model updates instead of raw data, making it suitable for applications like privacy-preserving image analysis in healthcare and decentralized AI tasks in IoT environments.

3.3.3 InceptionResNetV2

InceptionResNetV2 merges features from Google's Inception and ResNet architectures, combining Inception modules for multi-scale feature extraction with ResNet's residual connections for gradient flow and training stability. This hybrid design enhances both accuracy and efficiency in complex image recognition tasks. InceptionResNetV2's factorized convolutions and batch normalization optimize computational performance while maintaining high accuracy, making it suitable for demanding applications like image classification and object detection where robustness and speed are crucial.

3.4 InceptionV3 in Classification

We have used InceptionV3 for tasks such as analyzing screenshots of YouTube tabs. Its capability in extracting hierarchical features from images would facilitate accurate classification of tab categories like Home, Trending, or Subscriptions based on visual content. Moreover, InceptionV3's efficient design, including factorized convolutions and batch normalization, aligns well with the project's goal of preserving user privacy through federated learning. By processing data locally on client devices and transmitting only model updates, InceptionV3 enables collaborative model training while ensuring that sensitive user information remains secure. This makes InceptionV3 a suitable choice for enhancing both the classification accuracy of YouTube tabs and the privacy protection measures implemented in the project.

Chapter 4

Implementation

4.1 Introduction

In this chapter, we will delineate the dataset and workflow of our project, "YouTube Tab Classification and User Privacy using Federated Learning" explaining how privacy is being preserved using Federated Learning.

4.2 YouTube Tab Classification and User Privacy using Federated Learning

4.2.1 Dataset

In our project, we utilized approximately 2,700 YouTube screenshots categorized as productive or unproductive to train our model. Upon initiating the service, the system captures a screenshot of the YouTube tab every minute. Each screenshot is then analyzed by the local model to determine if it belongs to the productive or unproductive category.

Table 4.1: Dataset Distribution

Type of Image	No. of Images
Productive Images	1512
Unproductive Images	1181
Total Images	2693

4.3 Model Architecture

In our project, we have evaluated three models—InceptionV3, InceptionResNetV2, and a custom Convolutional Neural Network and chose the best model for classification of images and for maintaining the privacy of user federated learning is used.

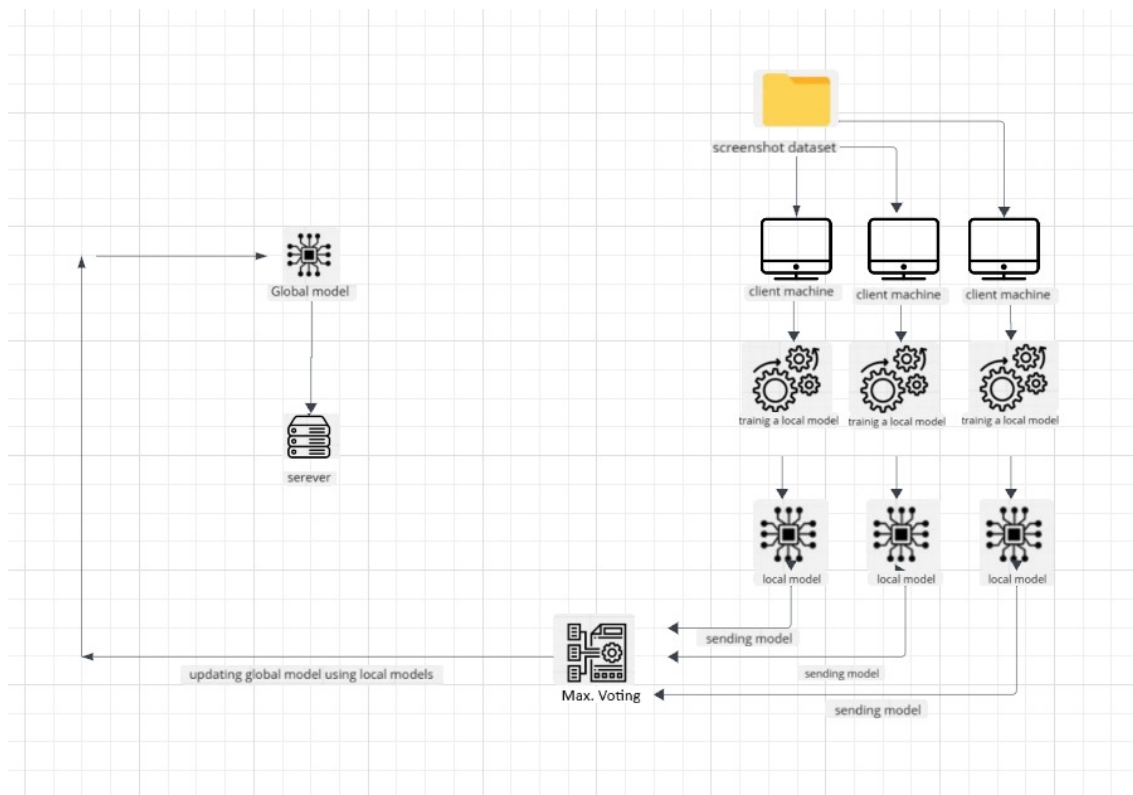


Figure 4.1: Workflow of Model

4.3.1 Model Comparison

We will be showing comparison between three models - InceptionV3, CNN and InceptionResNetV2 by measuring their performance on our dataset for classification into productive and unproductive images using different evaluation matrices.

- **Load, Split and Preprocess:** The dataset is loaded and split into training and validation. Further, it is preprocessed and configured for performance.
- **Initialization:** Three base models - InceptionV3, CNN and InceptionResNetV2 are initialized. The models are trained for training the dataset.
- **Evaluation and Comparison:** The models are then evaluated based on the evaluation matrices including accuracy, precision, recall, AUC and PRC.

```
1 title = data['title']
2 labels = ['educational', 'non-educational', 'entertainment', 'sports',
           'tutorial']
3 result = classifier(title, labels)
```

```
1 # Determine productivity based on classification
2 label = result['labels'][0]
3 if label in ['educational', 'tutorial']:
4     classification = 'productive'
5 else:
6     classification = 'unproductive'
7
8 return jsonify({'classification': classification})
```

After evaluation, the best model came out to be InceptionV3.

4.3.2 Train Global and Local Model

- **Global Model:** We have initialized an InceptionV3 model for binary classification, it freezes its feature extraction layers and saves the pretrained global model's state. This prepared

model is then saved as 'global-model.pth' for future use.

- **Local Model:** It trains an InceptionV3 model for binary classification on a local dataset of screenshots, then saves the updated model state. This local training adapts the pre-trained model to specific data while preserving data privacy.

```
1 # Load the global model
2 local_model = InceptionV3Model()
3 local_model.load_state_dict(torch.load('global_model.pth'))
```

4.3.3 Perform Detection

For Detecting whether the screenshot taken by Local Model is productive or unproductive, we have employed a YouTube extension called 'YouTube Tab Watcher'.

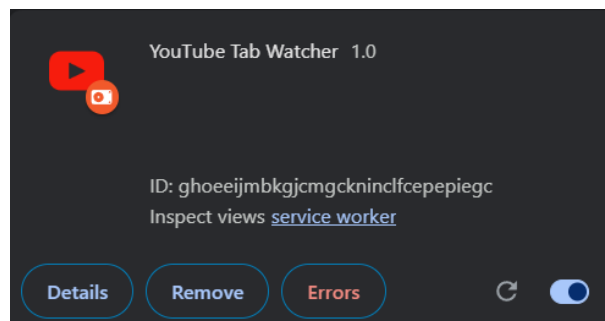


Figure 4.2: YouTube Tab Watcher

4.3.4 Creating Local Dataset

- **Extension:** YouTube extension is created named 'YouTube Tab Detector' which tracks YouTube tab activity in a browser, capturing details like the video's title. When a YouTube video is detected, it sends the title to the local API for classification as productive or unproductive.

- **Local Dataset Creation:** Based on the classification, it captures a screenshot and saves it in the corresponding folder via another API call. The process repeats every minute to continuously update and classify YouTube activity.

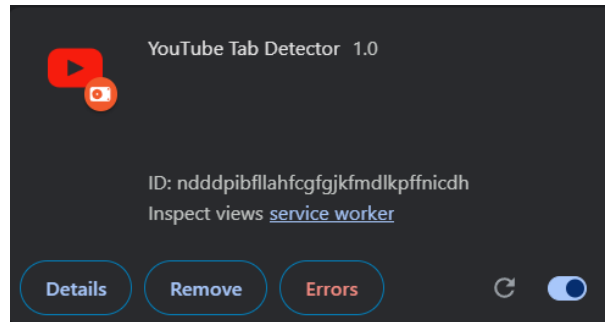


Figure 4.3: YouTube Tab Detector

4.3.5 Update Global Model

- **Background Work:** In background, code schedules and runs two Python scripts- client-side-back.py and aggregation-back.py, every minute using the schedule library, updating the global model using local model. It ensures continuous execution in an infinite loop with brief pauses to minimize CPU usage.

```
1  def run_client_side():
2  print("Running client-side.py")
3  subprocess.run(["python", "client-side-back.py"], check=True)
4
5  def run_aggregation():
6  print("Running aggregation.py")
7  subprocess.run(["python", "aggregation-back.py"], check=True)
```

- **Aggregation:** Aggregate updates from multiple local models using federated learning, and updates and saves the global model. This process ensures the global model benefits from distributed training while preserving data privacy.

```
1     # Load the state dicts for each local model
2 for i, model in enumerate(local_models):
3     model.load_state_dict(torch.load(local_model_paths[i]))
4
5 # Initialize a state dict to accumulate the average of the local
   models
6 global_dict = global_model.state_dict()
7 for k in global_dict.keys():
8     tensors = [local_models[i].state_dict()[k].float() for i in range(len(
        local_models))]
9     global_dict[k] = torch.stack(tensors, 0).mean(0)
```

4.4 Conclusion

In this chapter, we have outlined the implementation of our project focusing on YouTube Tab classification using best CNN model while ensuring the privacy preservation of client using Federated Learning.

The federated learning process aggregates insights from diverse client datasets to form a robust global model. This approach ensures data privacy by keeping raw data confidential within each client, enhancing model performance while maintaining confidentiality.

The resulting global model not only improves performance through collaborative knowledge extraction but also safeguards data privacy, making it ideal for distributed machine learning applications.

Chapter 5

Experiment and Result

In this chapter, we present results of our experiments.

To attain a better result, we trained three different CNN models to classify the YouTube activity of students and selected the best one for global model. We have used federated learning for privacy preservation of user and updating the global model.

The following experiments are depicted:

5.1 Experiment Setup

This project is implemented using VS Code and Google Colaboratory that grants free access for training and testing various machine learning models which are implemented using frameworks like tensorflow, keras, pytorch, and microframework like flask.

5.2 Model Evaluation

To access the performance of our model, we have used various evaluation metrics. We tested and validated the models- InceptionV3, CNN and InceptionResNetV2 on the dataset.

To evaluate the performance, the evaluation metrics includes:

- Accuracy: It is the ratio of correctly predicted instances to the total instances.

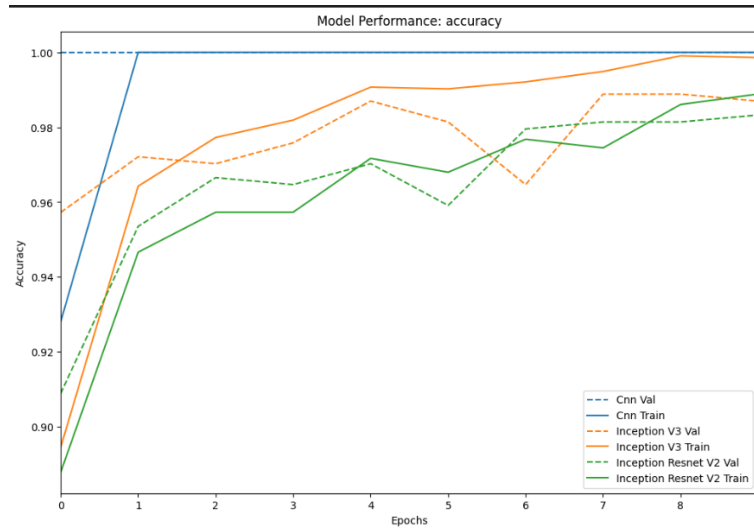


Figure 5.1: Accuracy

- Precision: It is the ratio of true positive predictions to the total predicted positives.

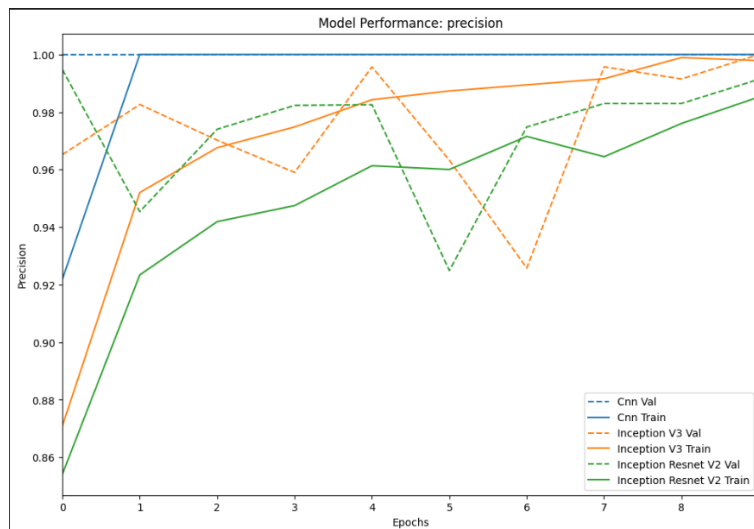


Figure 5.2: Precision

- Recall: It is the ratio of true positive predictions to the total actual positives.

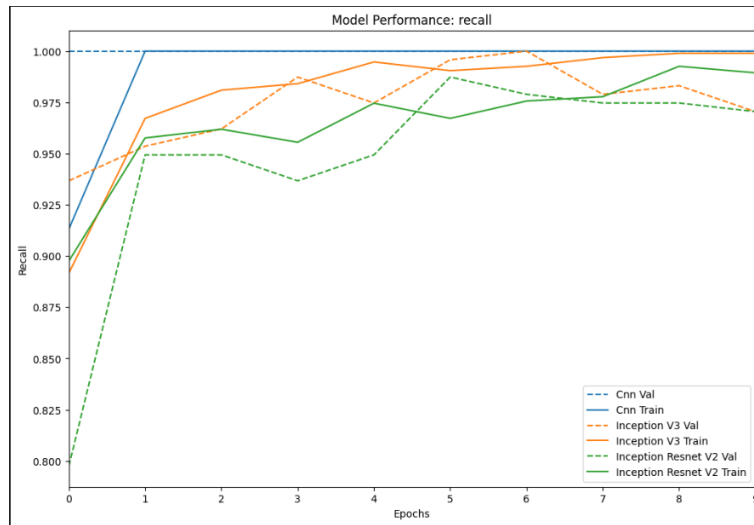


Figure 5.3: Recall

- Area under the curve: It indicates the trade-off between true positive and false positive rates.

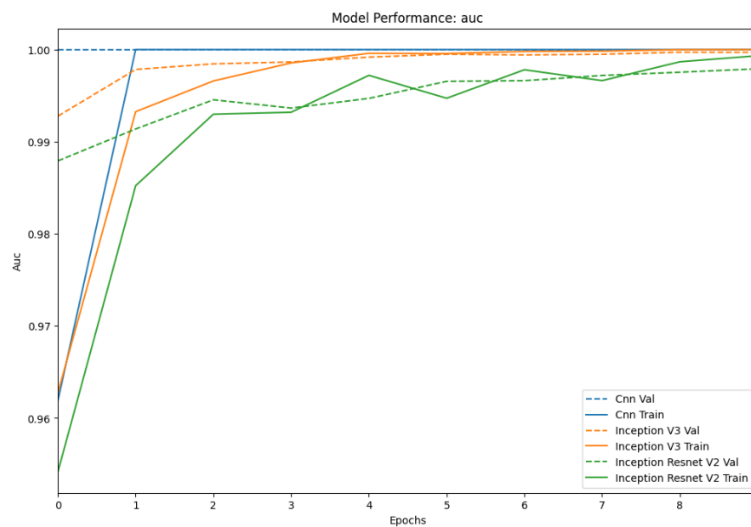


Figure 5.4: Area under curve

- Precision-Recall curve: It plots precision versus recall for different threshold values.

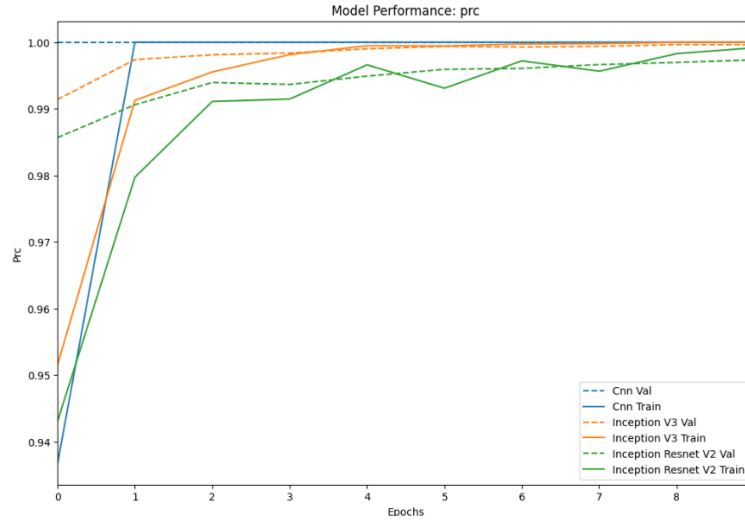


Figure 5.5: Precision-Recall Curve

Models	Accuracy	Precision	Recall	AUC	PRC
InceptionV3	0.987	1.0	0.9705	0.9997	0.9996
CNN	100	100	100	100	100
InceptionResNetV2	0.9833	0.9914	0.9705	0.9979	0.9974

Table 5.1: Model's performance metrics

5.2.1 Conclusion

By evaluating the performance of the models based on mentioned performance metrics, we conclude:

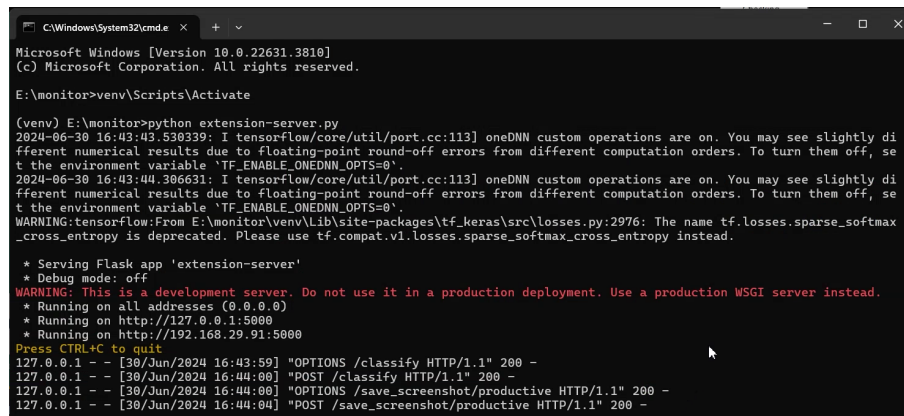
- InceptionV3 performs best compared to other 2.
- CNN is overfitting with the dataset.
- InceptionResNetV2 performs less efficiently than InceptionV3.

The best model came out to be InceptionV3 which is initialized as global model.

5.3 Result

5.3.1 Classification

Extension youtube tab watcher runs on the frontend which continuously takes screenshot of youtube tab every minute and sends it to global model. Local Model is the copy of Global model which runs in the background of extension and classifies the image as productive or unproductive and reflects it back on the extension.



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22631.3810]
(c) Microsoft Corporation. All rights reserved.

E:\monitor>venv\Scripts\Activate

(venv) E:\monitor>python extension-server.py
2024-06-30 16:43:43.528339: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-06-30 16:43:44.386631: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
WARNING:tensorflow:From E:\monitor\venv\Lib\site-packages\tf.keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

 * Serving Flask app 'extension-server'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.29.91:5000
Press CTRL+C to quit
127.0.0.1 - - [30/Jun/2024 16:43:59] "OPTIONS /classify HTTP/1.1" 200 -
127.0.0.1 - - [30/Jun/2024 16:44:00] "POST /classify HTTP/1.1" 200 -
127.0.0.1 - - [30/Jun/2024 16:44:00] "OPTIONS /save_screenshot/productive HTTP/1.1" 200 -
127.0.0.1 - - [30/Jun/2024 16:44:04] "POST /save_screenshot/productive HTTP/1.1" 200 -

```

Figure 5.6: Take screenshot and classify it

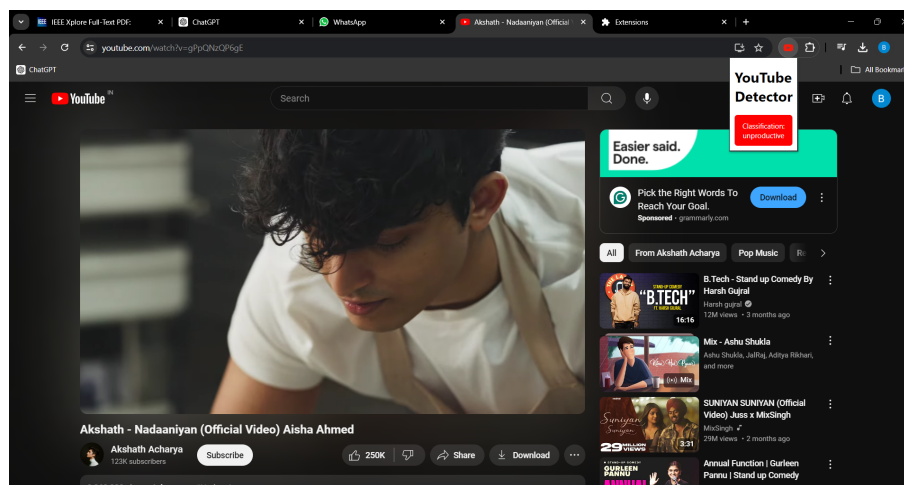


Figure 5.7: Classification of screenshot

5.3.2 Folder Creation

After taking the screenshot every minute, a folder named 'screenshot' is created in file manager by the extension youtube tab detector.

project_youtube_complete	29-06-2024 23:30	File folder
screenshots	29-06-2024 17:30	File folder
unproductive	29-06-2024 22:31	File folder

Figure 5.8: Screenshot folder

YouTube extension classifies the screenshot as productive or unproductive and save it in their respective folders.

Name	Date modified	Type	Size
productive	30-06-2024 16:38	File folder	
unproductive	29-06-2024 17:30	File folder	

Figure 5.9: Screenshot classified

Screenshot is saved after every minute.

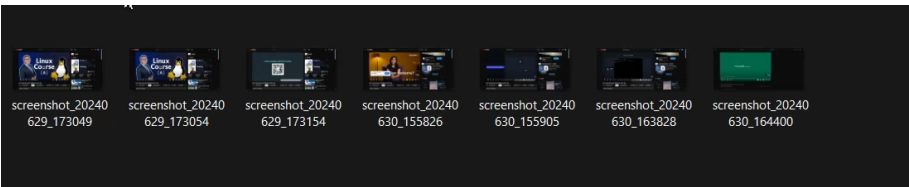
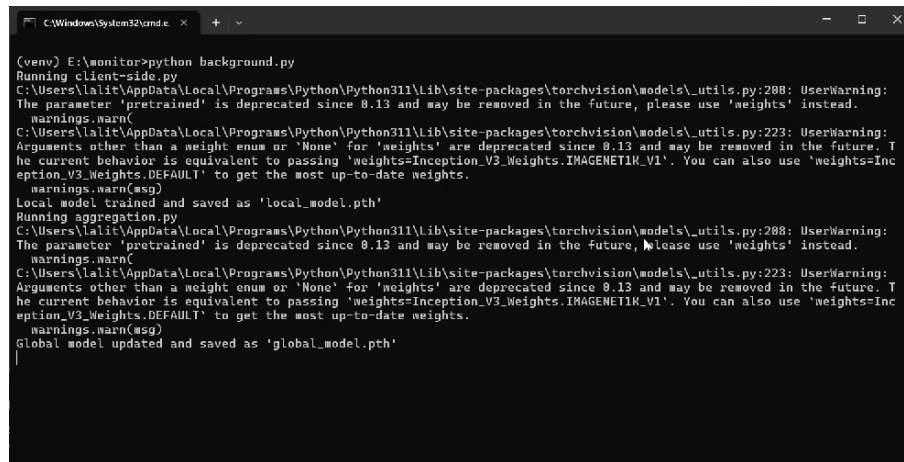


Figure 5.10: Screenshot saved

5.3.3 Update Global Model

In the background, the Client model and Global model run simultaneously at one-minute intervals. After the screenshot is taken, the Client model updates its weight to Global model. This also enables automated execution of client-side tasks and data aggregation at regular intervals.



```
(venv) E:\monitor>python background.py
Running client-side.py
C:\Users\lalit\AppData\Local\Programs\Python\Python311\Lib\site-packages\torchvision\models\_utils.py:208: UserWarning:
The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\lalit\AppData\Local\Programs\Python\Python311\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning:
Arguments other than a weight enum or 'None' for 'weights' are deprecated since 0.13 and may be removed in the future. T
he current behavior is equivalent to passing 'weights=Inception_V3_Weights.IMAGENET1K_V1'. You can also use 'weights=Inc
eption_V3_Weights.DEFAULT' to get the most up-to-date weights.
  warnings.warn(msg)
Local model trained and saved as 'local_model.pth'
Running aggregation.py
C:\Users\lalit\AppData\Local\Programs\Python\Python311\Lib\site-packages\torchvision\models\_utils.py:208: UserWarning:
The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
C:\Users\lalit\AppData\Local\Programs\Python\Python311\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning:
Arguments other than a weight enum or 'None' for 'weights' are deprecated since 0.13 and may be removed in the future. T
he current behavior is equivalent to passing 'weights=Inception_V3_Weights.IMAGENET1K_V1'. You can also use 'weights=Inc
eption_V3_Weights.DEFAULT' to get the most up-to-date weights.
  warnings.warn(msg)
Global model updated and saved as 'global_model.pth'
```

Figure 5.11: Update Global Model

Chapter 6

Conclusion & Future Work

6.1 Conclusion

In this project, we have successfully implemented a system for classifying YouTube tabs using InceptionV3, a powerful Convolutional Neural Network, while ensuring user privacy through Federated Learning.

By leveraging InceptionV3's capabilities in image recognition, we achieved high accuracy in categorizing tabs based on visual content.

Federated Learning proved effective in preserving user privacy by enabling model training on decentralized data sources without compromising sensitive user information. This approach not only enhances the accuracy of content of our systems but also sets a precedent for privacy-conscious machine learning applications in digital platforms.

Moreover, Federated Learning has proven instrumental in addressing privacy concerns by allowing model training on local devices while only exchanging model updates with a central server. This decentralized approach ensures that user data remains private and secure, mitigating risks associated with centralized data storage and processing.

By adopting Federated Learning, we not only uphold user privacy but also pave the way for scalable and ethical machine learning solutions in digital platforms.

6.2 Future Work

There is much scope of further advancements in the project developed, some of which include:

1. Increase the size of dataset to make the model work more accurately: Collect additional data from various sources to ensure the dataset is diverse and representative. Implement techniques to handle class imbalance if any class is underrepresented in the dataset.
2. Make use of the achieved classification to make conclusions about the user: We can perform sentiment analysis, make youtube recommendation system for user etc. Utilize the classification results to perform sentiment analysis on user interactions, feedback or other text data. Collaborative filtering, content-bases filtering or hybrid recommendation algorithms can be implemented to enhance the recommendation accuracy.
3. Accomodate screens of various devices to enhance its applicability in the real-world: Usability and accessibility across various devices can be enhanced by creating a responsive design, cross-platform applications and conducting user experience enhancements.

Bibliography

- [1] N. Hasan B. J. Ferdosi, M. Sadi and M. A. Rahman. Tracking digital device utilization from screenshot analysis using deep learning, proc. int. conf. data sci. appl. (icdsa), vol. 1, pp. 661-670, 2023.
- [2] <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10196311>. ‘consumption of social media and academic performance: A cross-sectional survey of perception of students in kp universities, global mass commun. rev., vol. 4, pp. 57-71, 2020.
- [3] B. Imler and M. Eichelberger. Using screen capture to study user research behavior, library hi tech, vol. 29, no. 3, pp. 446-454, 2011.
- [4] J.Goet. ‘impact of social media on academic performance of students, kic int. j. social sci. manage., vol. 1, no. 1, pp. 35-42, dec. 2022.
- [5] P. Krieter and A. Breiter. Track every move of your students: Log files for learning analytics from mobile screen recordings, proc. die 16. e-learning fachtagung informatik (delfi), pp. 1-12, 2018.
- [6] Dhurgham Hassan Mahloul and Mohammed Hamzah Abed. A comprehensive survey on federated learning: Concept and applications, proc. acm sigsac conf. comput. commun. secur., pp. 1175-1191, oct. 2017.
- [7] Zeyi Wen Qinbin Li and Bo Li. A survey on federated learning: challenges and applications, manuscript. preliminary version, vol. 78, no. 110, pp. 1-108, 1998.

- [8] H. S. Ullah S. I. U. Rehman and A. Akhtar. ‘consumption of social media and academic performance: A cross-sectional survey of perception of students in kp universities, global mass commun. rev., vol. 5, no. 4, pp. 57-71, dec. 2020.
- [9] I. K. Sohail and N. A. Nabaz. The influence of social media on student’s academic performance: A case study of lebanese french university, mod. manag. theory pract., vol. 25, no. 2, pp. 117-127, 2019.

Appendices

A. Background

```
1  def run_client_side():
2      print("Running client-side.py")
3      subprocess.run(["python", "client-side-back.py"], check=True)
4
5  def run_aggregation():
6      print("Running aggregation.py")
7      subprocess.run(["python", "aggregation-back.py"], check=True)
8
9  # Schedule the functions
10 schedule.every().minute.do(run_client_side)
11 schedule.every().minute.do(run_aggregation)
12
13 # Run the scheduler in an infinite loop
14 while True:
15     schedule.run_pending()
16     time.sleep(1) # Sleep for a short time to prevent high CPU
                   usage
```

B. Server Side

```
1 class InceptionV3Model(nn.Module):
2     def __init__(self):
3         super(InceptionV3Model, self).__init__()
4         # Load the pre-trained Inception V3 model
5         self.model = models.inception_v3(pretrained=True)
6
7         # Freeze all the parameters in the feature extraction
            layers
8         for param in self.model.parameters():
9             param.requires_grad = False
10
11        # Replace the classifier part of Inception V3
12        num_features = self.model.fc.in_features
13        self.model.fc = nn.Linear(num_features, 1) # Adapted for binary
            classification
14
15    def forward(self, x):
16        # Inception V3's forward method may return auxiliary
            outputs
17        # when training, which we do not need during inference
18        if self.model.training:
19            x, _ = self.model(x)
20        else:
21            x = self.model(x)
22        return torch.sigmoid(x)
23
24 # Initialize the global model
25 global_model = InceptionV3Model()
```

```
26
27 # Save the global model
28 torch.save(global_model.state_dict(), 'global_model.pth')
29 print("Global model created and saved as 'global_model.pth'")
```

C. Client Side

```
1 # Load the global model
2 local_model = InceptionV3Model()
3 local_model.load_state_dict(torch.load('global_model.pth'))
4
5 # Define the data transformations
6 transform = transforms.Compose([
7     transforms.Resize((299, 299)), # Update size for Inception V3
8     transforms.ToTensor(),
9     transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) #
    Standard normalization for ImageNet
10 ])
11
12 # Load the local dataset
13 train_data = datasets.ImageFolder('screenshots/', transform=transform)
14 train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
15
16 # Define the loss and optimizer
17 criterion = nn.BCEWithLogitsLoss() # Use BCEWithLogitsLoss
18 optimizer = optim.Adam(local_model.parameters(), lr=0.001)
19
20 # Train the model locally
21 local_model.train()
```

```
22 for epoch in range(5): # Local training for 5 epochs
23     for images, labels in train_loader:
24         optimizer.zero_grad()
25         outputs = local_model(images)
26         loss = criterion(outputs, labels.float().unsqueeze(1))
27         loss.backward()
28         optimizer.step()
29
30 # Save the updated local model
31 torch.save(local_model.state_dict(), 'local_model.pth')
32
33 print("Local model trained and saved as 'local_model.pth'")
```

=