Advancing Pneumonia Detection with Federated Learning and Chest X-Ray Analysis

Abstract

This project focuses on addressing the critical healthcare challenge of pneumonia detection through the analysis of chest X-ray images.The primary goal of this study is to develop a robust and efficient predictive model that can accurately classify chest X-ray images into two categories: normal and pneumonia.The project began with a comparative analysis of various machine learning and deep learning algorithms to identify the most effective model for the task. After rigorous experimentation and evaluation, the best-performing model was selected based on its accuracy and reliability. Building upon this foundation, federated learning was incorporated to leverage distributed training across multiple clients without compromising the privacy of sensitive medical data.

Keywords: Pneumonia detection, chest X-ray, machine learning, deep learning, binary classification, Logistic Regression, Decision Tree, Random Forest, Neural Network, Convolutional Neural Networks (CNNs), InceptionV3, Federated learning, Distributed Training, Class imbalance handling, Data augmentation, Flower-AI

Introduction

Problem Statement: Detect the presence of pneumonia in a chest X-ray image by incorporating Federated Learning in the pipeline.

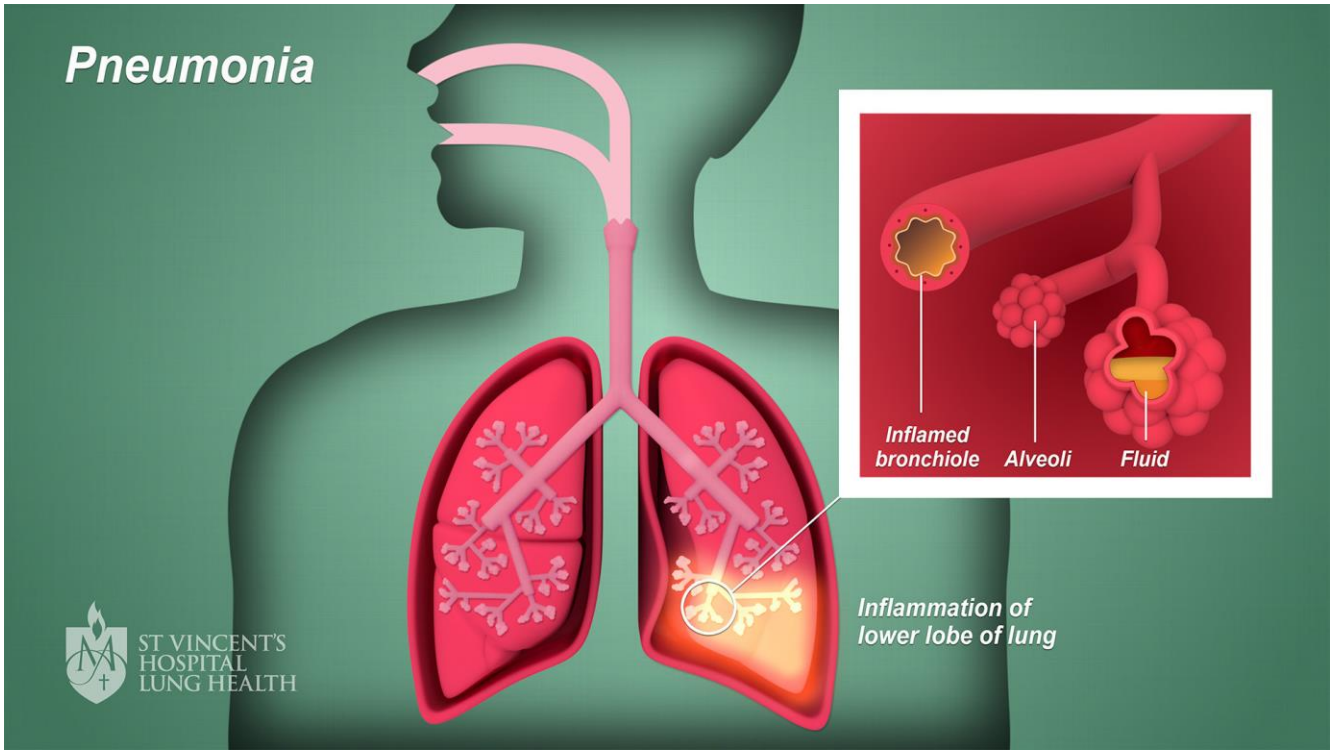
Pneumonia is a severe respiratory condition caused by infections that can lead to life-threatening complications if not detected early. Chest X-rays are a primary diagnostic tool for pneumonia, but their interpretation requires expertise, which may not always be available. This project aims to address this challenge by developing a predictive model that classifies X-ray images into two categories: Pneumonia and Normal.

The objective is to develop an accurate and efficient binary classification model for detecting pneumonia using chest X-rays while leveraging federated learning to maintain data privacy.

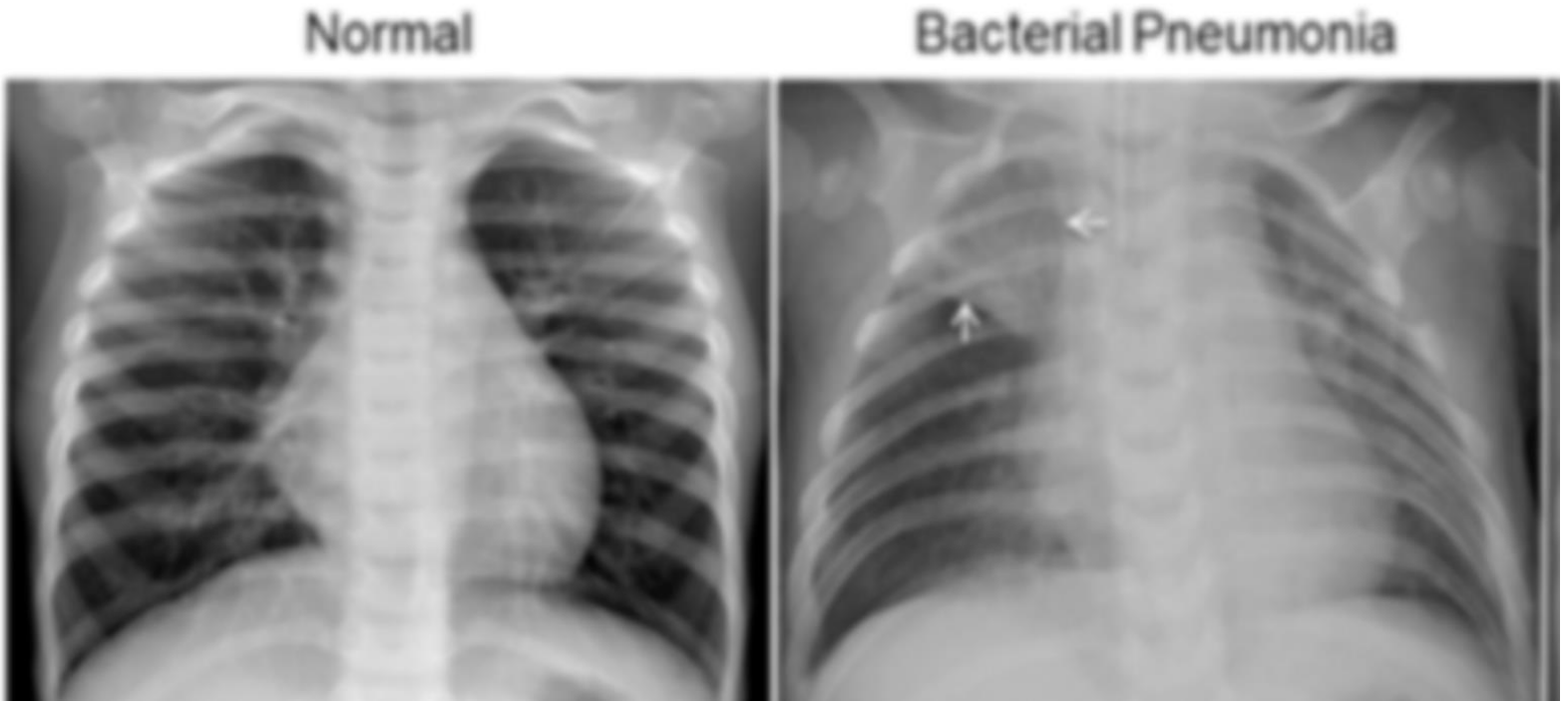
What is Pneumonia?

Pneumonia is a respiratory infection that primarily affects the lungs, often caused by various microorganisms, including bacteria, viruses, and fungi. The infection triggers **inflammation**, which is the body's natural immune response to harmful stimuli, resulting in redness, swelling, and increased blood flow to the affected area. In pneumonia, this inflammation affects the tiny air sacs in the lungs, called alveoli, which are responsible for the crucial exchange of oxygen and carbon dioxide. The inflamed air sacs become filled with pus and other fluids, leading to symptoms such as persistent cough, difficulty breathing, chest pain, and fever. This disruption in lung function can range from mild to severe, depending on the underlying cause and the individual's health condition.

Pneumonia can range in seriousness from mild to life-threatening. It is most serious for infants and young children, people older than age 65, and people with health problems or weakened immune systems.



How To Distinguish between a Pneumonia and a Normal chest X-ray?



Looking at the X-ray picture of Pneumonia infected person we can notice, In the middle panel,a cloudy or hazy area in the right upper part of the lung which is pointed out by white arrows.

What it means: This cloudy area is a sign of something not right. It's like a shadow or fog in a specific part of the lung and this is the indication of presence of pneumonia Disease in Lungs.

Dataset Description

The dataset used in this project consists of chest X-ray images classified into two categories: **Normal** and **Pneumonia**. These images are organized into three distinct subsets: **Training**, **Validation**, and **Testing**, to facilitate the development and evaluation of the predictive model.

Normal: Images depicting healthy lungs without any signs of infection.

Pneumonia: Images indicating the presence of pneumonia, caused by bacterial or viral infection.

Structure:

Training Set: Used to train the model, containing the majority of the dataset images. It includes a balanced mix of Normal and Pneumonia images to ensure effective learning.

Validation Set: Used to fine-tune model hyperparameters and prevent overfitting. This set contains a smaller number of images, providing real-time feedback on model performance during training.

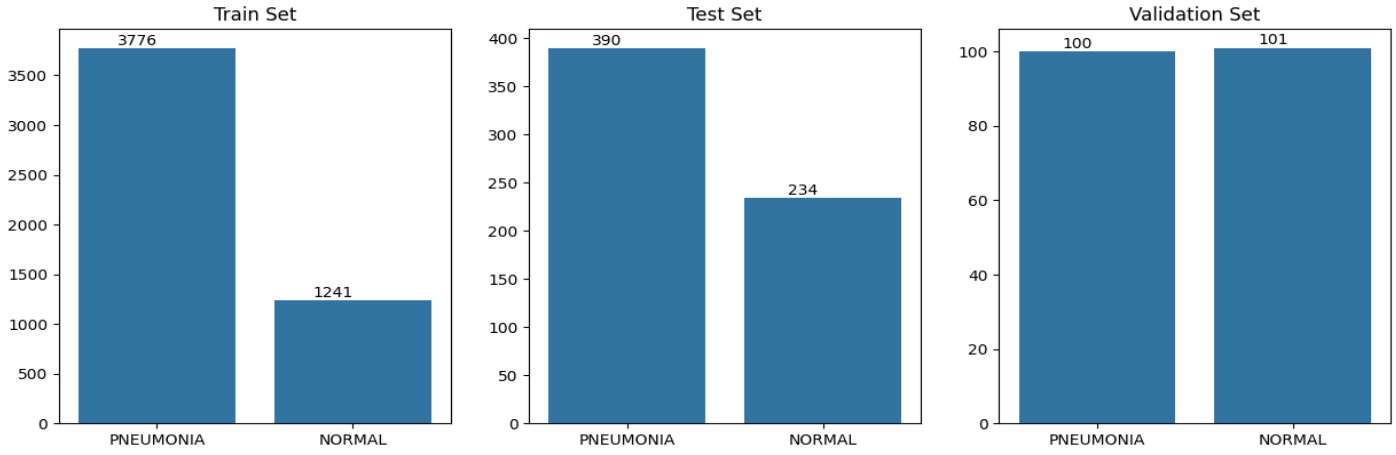
Testing Set: Held out for the final evaluation of the model's performance, ensuring an unbiased assessment of its ability to generalize to unseen data.

Class Distribution:

The dataset contains a higher proportion of pneumonia cases compared to normal cases, reflecting the real-world prevalence of the disease. Class weights are applied during training to address this imbalance.

Dataset Source:

The dataset is sourced from publicly available medical imaging datasets on **KAGGLE,** ensuring diversity in imaging techniques and patient demographics.



Models Utilized for Pneumonia Detection

To address the binary classification problem of distinguishing between normal and pneumonia chest X-rays, a series of machine learning and deep learning models were employed. The aim was to evaluate and compare the performance of these models to identify the most suitable approach for accurate pneumonia detection. Each model was trained, validated, and tested on the dataset described earlier. The models range from classical machine learning algorithms to state-of-the-art deep learning architectures, followed by the integration of federated learning for enhanced data privacy and scalability.

Logistic Regression

Logistic Regression is a foundational machine learning algorithm widely used for binary classification tasks. In this project, it was employed as an initial model to classify chest X-ray images into two categories: Normal and Pneumonia. Its simplicity and interpretability make it an excellent starting point for understanding the dataset and establishing a baseline for performance.

How Logistic Regression Works

Logistic Regression predicts the probability of an outcome based on input features. For binary classification, it outputs a value between 0 and 1, representing the likelihood of belonging to a particular class. This probability is then converted into one of two categories (e.g., Normal or Pneumonia) using a threshold value, typically set at 0.5. The algorithm uses a logistic function (also called a sigmoid function) to map raw predictions into probabilities. Logistic Regression learns from the training data by identifying relationships between the input features (e.g., pixel values or other numerical representations of X-ray images) and their corresponding labels.

Model Accuracy : 81.25%

Decision Tree

A Decision Tree is another popular algorithm used for classification tasks. It models the decision-making process by splitting the data into distinct subsets based on certain feature values. In this project, a Decision Tree classifier was used to predict the presence of pneumonia in chest X-ray images. The algorithm works by creating a tree-like structure, where each internal node represents a feature (or attribute), each branch represents a decision rule, and each leaf node represents an outcome (i.e., the predicted class label).

How Decision Tree Works

A Decision Tree classifier constructs a tree by recursively partitioning the feature space, choosing the most significant features at each step. The main goal is to create splits that best separate the classes (Normal or Pneumonia) in the data. At each decision node, the algorithm evaluates which feature and threshold will give the best separation based on some criterion, such as Gini impurity or Information Gain.

The process of building the tree involves:

Splitting: The algorithm splits the data based on different feature values (e.g., pixel intensities, color histograms, or other extracted features from the X-ray images). The aim is to reduce uncertainty and increase homogeneity within each subset.

Stopping Criteria: The algorithm continues to split the data recursively until a stopping condition is met. This could be a maximum depth of the tree, a minimum number of samples in a node, or when further splits no longer significantly improve the model.

Prediction: Once the tree is built, new data points (X-ray images) are passed through the tree, following the decision rules from the root node to the leaf node. The leaf node will provide the predicted class label (Normal or Pneumonia).

Model Accuracy : 77.24%

Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to make more robust predictions. It is one of the most powerful and widely used algorithms for classification and regression tasks. Random Forest works by training a large number of decision trees, each trained on a random subset of the training data, and making predictions by averaging the results of individual trees (for regression tasks) or using a majority voting system (for classification tasks).

In this project, Random Forest was used as an enhancement over individual decision trees to improve the accuracy and generalizability of the model for detecting pneumonia in chest X-ray images. By leveraging multiple trees, Random Forest overcomes the limitations of a single decision tree, such as overfitting and instability, and provides more accurate and stable predictions.

How Random Forest Works

Bootstrap Aggregating (Bagging): Random Forest operates by using a technique called bagging (Bootstrap Aggregating). This involves creating multiple subsets of the training data through random sampling with replacement. Each tree in the forest is trained on a different subset, which helps to reduce overfitting and improve the model's ability to generalize to new data.

Random Feature Selection: In addition to sampling the data, Random Forest also introduces randomness in the selection of features used for splitting the nodes in each tree. Instead of considering all features, a random subset of features is selected at each node to make the split. This increases the diversity among the trees and helps reduce the correlation between them, improving overall model performance.

Training Multiple Decision Trees: Random Forest creates many decision trees (often hundreds or thousands), each trained on a different bootstrapped subset of the training data with random feature selections. As a result, each tree is slightly different from the others, which helps capture different patterns in the data.

Making Predictions: Once the forest of decision trees is trained, predictions are made by passing the input data through each tree. For classification tasks, each tree votes for a class label, and the class with the most votes becomes the final prediction. For regression tasks, the predictions are averaged across all the trees to give the final output.

Model Accuracy : 81.73% (5 Decision Trees implemented)

Hyperparameter Tuning for Random Forest Using Grid Search

Grid Search systematically evaluates all combinations of a predefined set of hyperparameters. It performs exhaustive search across a grid of hyperparameter values and returns the combination that provides the best performance on the validation dataset.

Hyperparameter Tuned in Random Forest Using Grid Search

n_estimators (Number of Trees):

This hyperparameter defines how many trees should be included in the Random Forest.

A higher number of trees generally improves the model's performance, but at the cost of increased computational time and memory usage.

How Grid Search Works for Hyperparameter Tuning in Random Forest

Define a Hyperparameter Grid: First, we define the hyperparameter grid, specifying a range of values for each hyperparameter that we want to tune. For instance, we are testing different numbers of trees in our case.

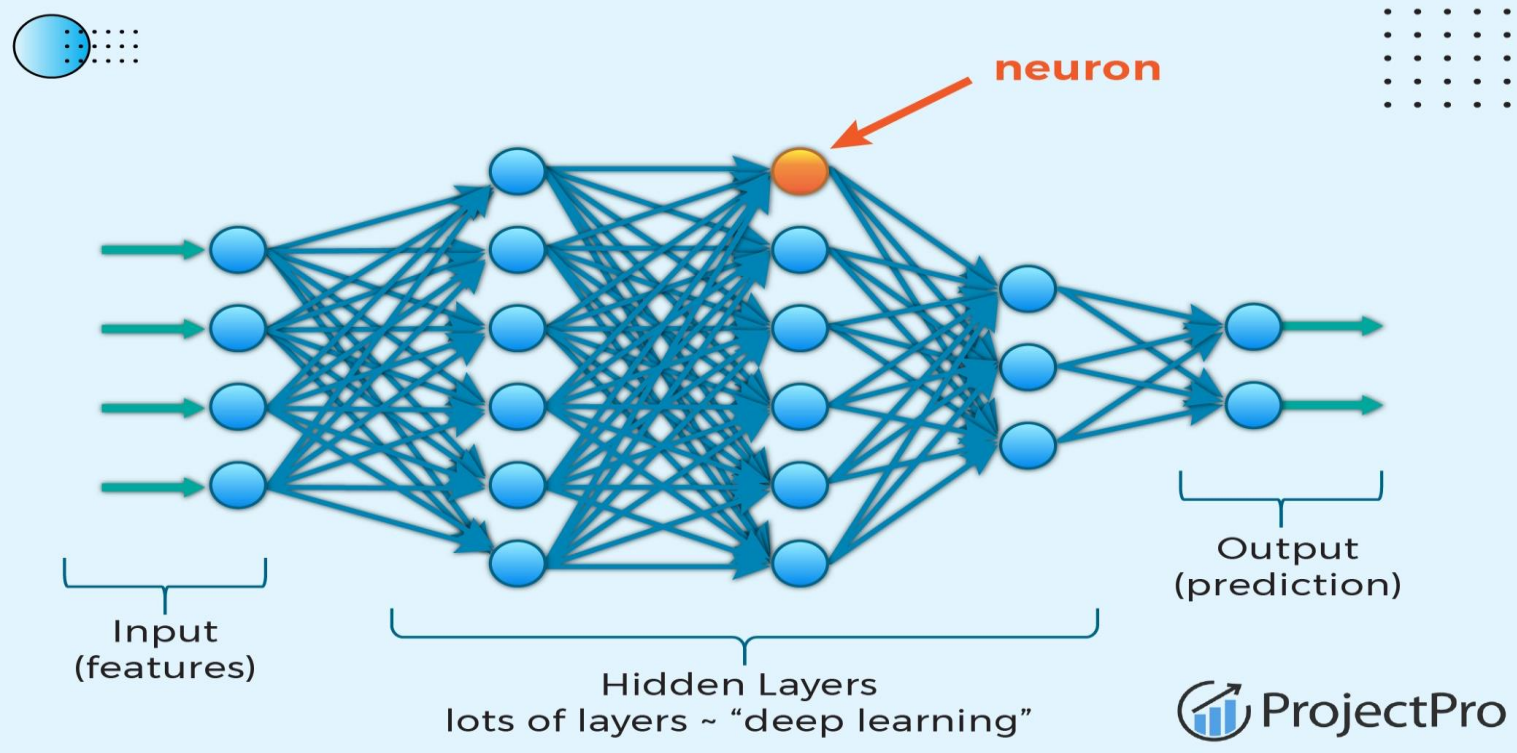
Cross-Validation: Grid search performs cross-validation to evaluate each hyperparameter combination. Cross-validation involves splitting the training data into multiple subsets (folds), training the model on one subset and validating it on the others. This helps ensure that the model generalizes well to new, unseen data.

Training and Evaluation: For each combination of hyperparameters, grid search trains a Random Forest model and evaluates it on the validation dataset. The performance is typically evaluated using metrics such as accuracy, F1-score, or ROC-AUC.

Best Combination: After evaluating all possible combinations of hyperparameters, the grid search selects the set that results in the best performance according to the chosen evaluation metric.

Model Accuracy : 85.73% (500 Decision Trees implemented)

Transitioning to Deep Learning: Unlocking Potential Beyond Traditional Models



Having explored traditional models like Logistic Regression, Decision Trees, and Random Forests for our task, the journey now unfolds into the realm of Deep Learning, specifically Neural Networks. Deep Learning introduces architectures that can learn more complex patterns and relationships within complex datasets.

Now, as we switch from regular models to Deep Learning, it's like moving from traditional tools to a supercharged detective for our job of telling if someone has pneumonia from their X-ray. Think of it as upgrading from a magnifying glass to a powerful detective gadget!

Why Deep Learning?

Because this new detective, inspired by how our brain works, is super good at spotting tricky details in X-ray pictures. It's like having a detective that can see and understand things that regular tools might miss.

Pixels Matter:

In our journey, we want not just to be right but to really understand the details in those pixelated X-ray pictures. Deep Learning helps us look at each tiny square (pixel) to figure out the story it's telling us about pneumonia.

First Baseline Simple Neural Network Architecture

We've embarked on constructing a straightforward neural network architecture using the Keras library. The model begins with a Flatten layer, transforming our 180x180x3 input images into a flat vector. Following this, we introduce two dense (fully connected) layers, each with rectified linear unit (ReLU) activation functions, promoting non-linearity in the model.

The final layer, a Dense layer with a single neuron and a **sigmoid activation function**, serves as the output layer. The sigmoid function is apt for **binary classification** tasks, making our neural network well-suited for distinguishing between normal and pneumonia X-ray images.

Model Accuracy : 80.44%

Pneumonia Detection with Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models primarily used for analyzing visual data, such as images or videos. They are designed to automatically and adaptively learn spatial hierarchies of features by using layers of convolutions, pooling, and fully connected layers.

Convolutional Layer: This layer applies convolution operations to the input data (such as an image) using filters (kernels). Each filter detects specific features (edges, textures, shapes) in the image.

Activation Function (ReLU): After convolution, an activation function, typically the Rectified Linear Unit (ReLU), is applied to introduce non-linearity. It helps the model learn complex patterns.

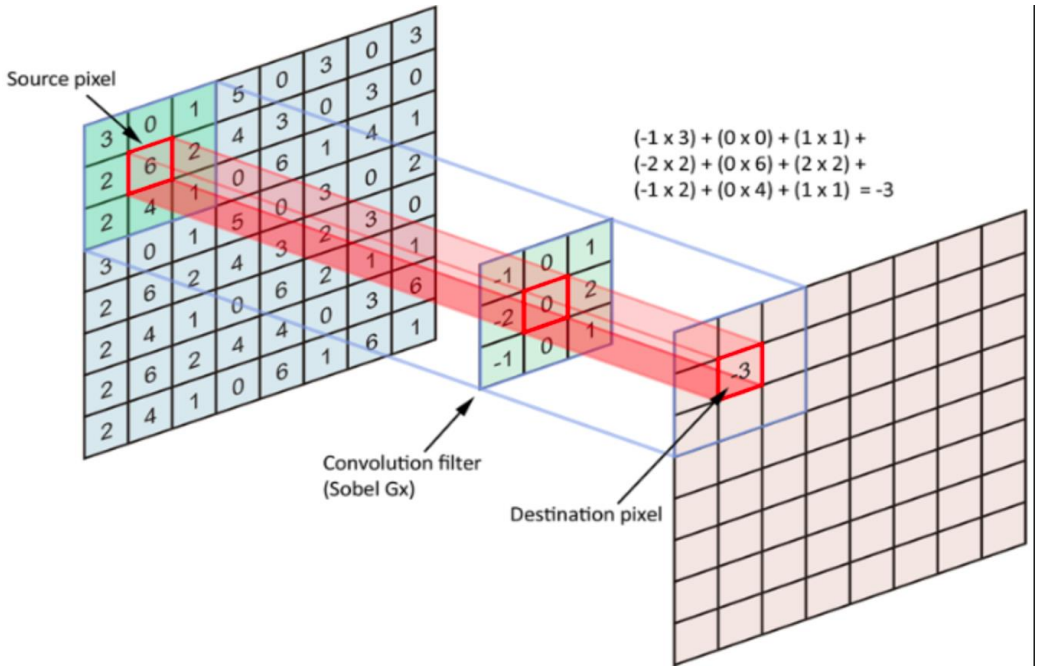
Pooling Layer: Pooling reduces the dimensionality of the data by selecting the maximum or average value from a subset of the input, helping to reduce computation and overfitting.

Fully Connected Layer: After passing through convolution and pooling layers, the output is flattened and connected to fully connected layers that predict the final output, like classification labels.

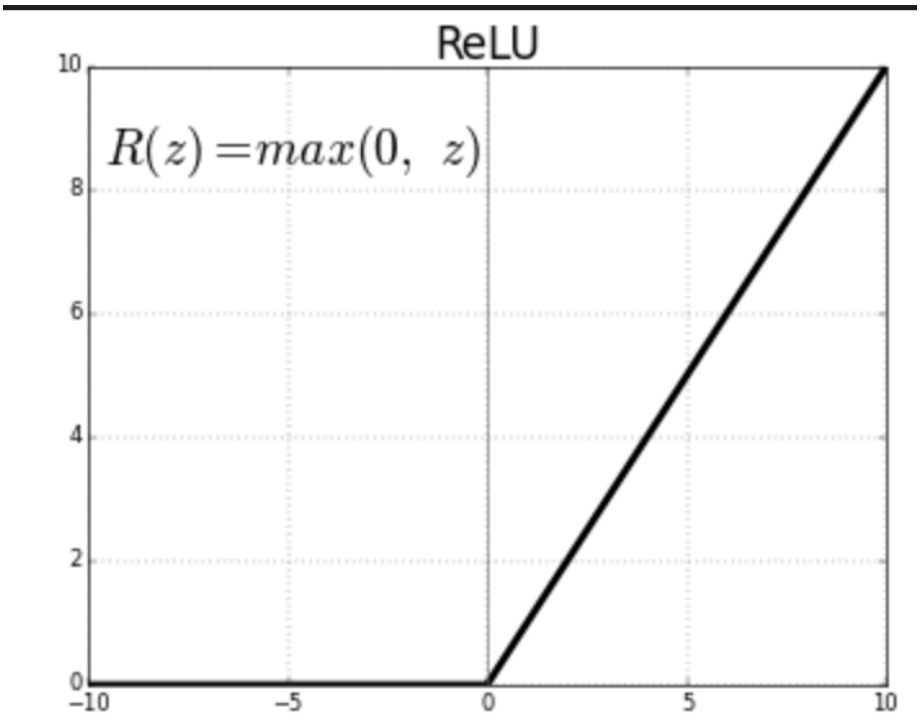
Output Layer: The final layer produces the prediction, such as classifying an image into different categories.

Convolution process

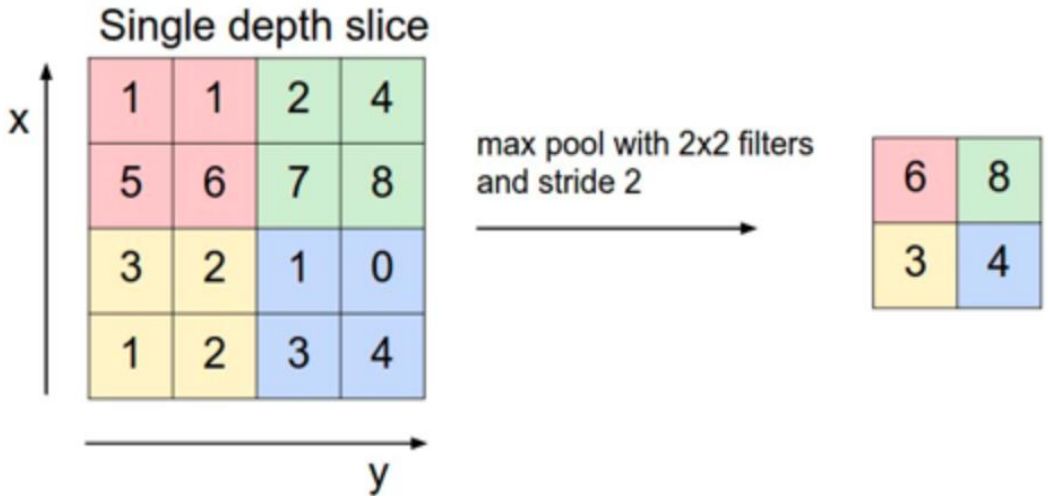
- **Using convolution filters with different dimensions or values results in differents features extracted**



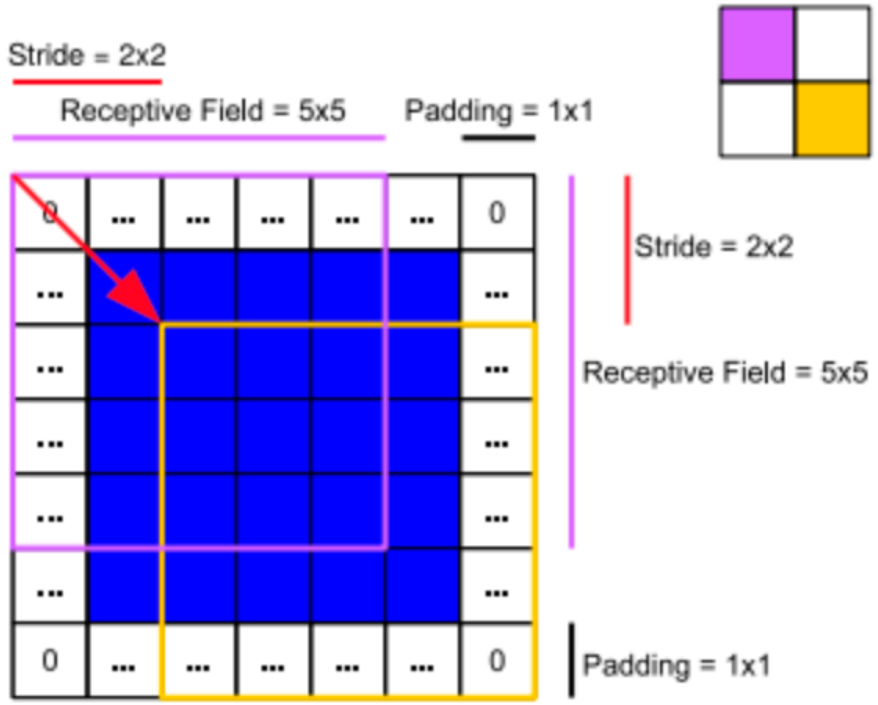
- **Features are then detected using the reLu activation on each destination pixel**



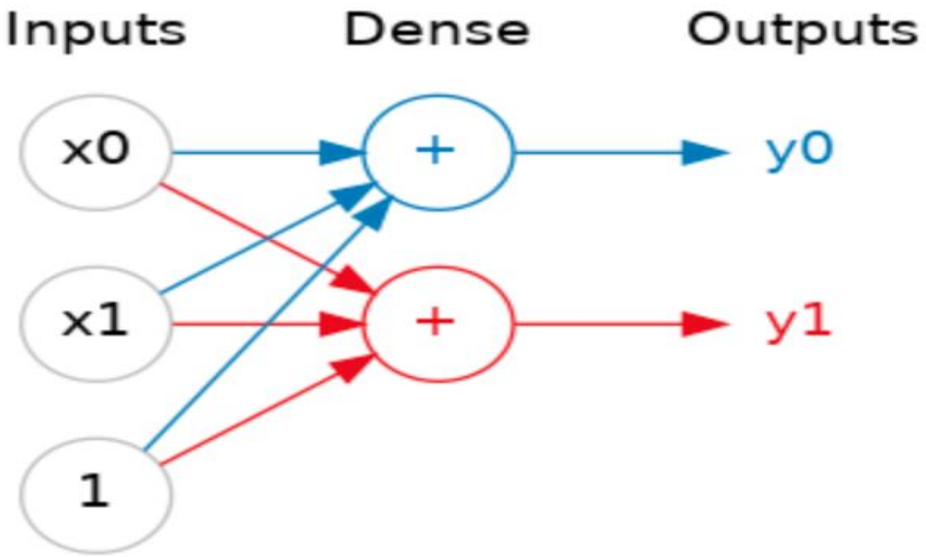
- **Features are the enhanced with MaxPool layers**



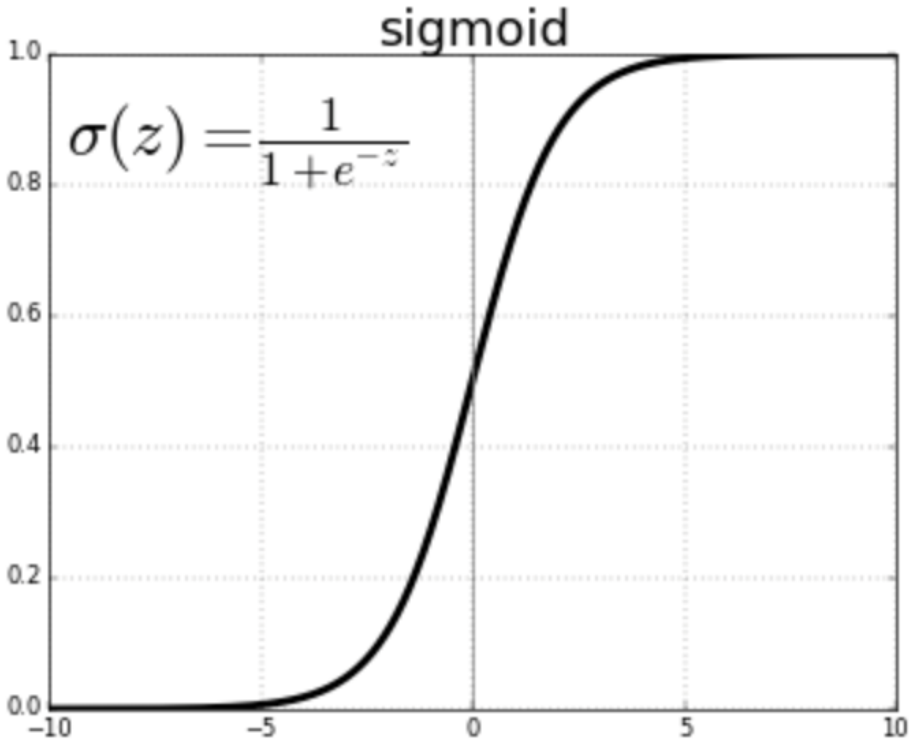
- **The stride parameters determines the distance between each filters. The padding one determines if we ignore the borderline pixels or not (adding zeros helps the neural network to get information on the border)**



- **The outputs are then concatened in Dense layers**



- **By using a sigmoid activation, the neural network determines which class the image belongs to**



Model Accuracy : 86.85%

Transfer Learning

The next approach we employed is **Transfer Learning**, a powerful technique that leverages pre-trained models to enhance performance and reduce training time, especially when dealing with limited data. Transfer learning involves using a model that has already been trained on a large and diverse dataset as a feature extractor. In our case, we utilized **InceptionV3**, a state-of-the-art deep learning model available in the Keras library.

InceptionV3 is a robust convolutional neural network that has been pre-trained on the **ImageNet** dataset, which contains millions of images across thousands of categories. By using this pre-trained model, we capitalize on its ability to extract complex and high-level features learned during its extensive training process.

For our task, we modified the InceptionV3 model to suit our pneumonia detection objective. Specifically, we set the `include_top` parameter to `False`, which removes the model's original classification head responsible for assigning categories in the ImageNet dataset. This step ensures that only the pre-trained feature extraction layers are retained.

Next, we added custom layers tailored to our binary classification problem. These layers, including a dense layer with a sigmoid activation function, were appended to the base model to handle our specific task of distinguishing between normal and pneumonia-affected chest X-rays. During training, we froze the pre-trained layers to retain their learned weights, while allowing the newly added layers to learn and adapt to our dataset.

This approach not only reduces the computational cost but also accelerates the training process while maintaining high accuracy. Transfer learning with InceptionV3 proved to be a highly effective strategy for our pneumonia detection project.

Model Accuracy : 87.5%

Fine Tuning

Our final approach is **Fine-Tuning**, a technique used to further enhance the performance of a pre-trained model by allowing specific layers to adapt to the new dataset. In the previous approach, we utilized transfer learning by freezing all the layers of the pre-trained model, preserving the weights that were learned during its training on the **ImageNet** dataset. This ensured that the model retained its ability to extract general-purpose features while the newly added layers learned to classify our dataset.

In fine-tuning, we take this process a step further by **unfreezing the last few layers** of the pre-trained model. This allows these layers to adjust their weights based on the patterns and features specific to our dataset, while the earlier layers remain frozen. The rationale behind this approach is that the initial layers of a convolutional neural network generally learn low-level features, such as edges and textures, which are universal across most image datasets. On the other hand, the deeper layers capture more complex, task-specific features, making them suitable candidates for fine-tuning.

By fine-tuning these layers, we optimize the model to better suit our pneumonia detection task while still leveraging the powerful feature extraction capabilities of the pre-trained model. This approach strikes a balance between retaining the knowledge of the original training and adapting the model to our specific data, further improving its classification accuracy.

Model Accuracy : 88.30%

Federated Learning Architecture: Extending the Best Model for Decentralized Training

After identifying the best-performing model for pneumonia detection through rigorous evaluation of various machine learning and deep learning approaches, the next step was to implement a **federated learning (FL) architecture**.

Federated learning offers a decentralized training paradigm where the model is trained collaboratively across multiple clients without centralizing the data. This approach enhances **data privacy** by ensuring sensitive medical data, such as chest X-ray images, remains stored locally on each client device or institution.

The selected model, proven to deliver high accuracy and robust performance, was adapted for the federated learning setup. This involved:

Distributing the Training Process: The global model is initialized on the server and distributed to all participating clients.

Local Training: Each client trains the model on its local dataset, ensuring privacy preservation and compliance with data protection regulations.

Global Aggregation: The locally updated models are sent back to the server, where their weights aggregated using techniques like Federated Averaging (FedAvg) to update the global model.

Iterative Optimization: This process repeats across multiple communication rounds, refining the global model with contributions from all clients.

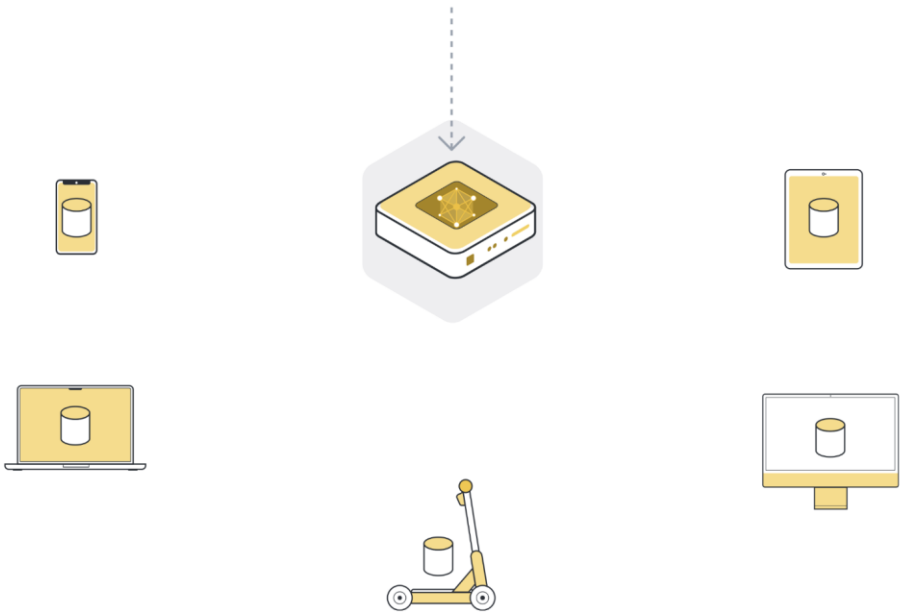
What is Federated Learning?

Federated Learning (FL) is a distributed machine learning approach that enables multiple devices or organizations (often referred to as clients) to collaboratively train a global model without sharing their raw data. This paradigm is particularly well-suited for privacy-sensitive applications, such as healthcare, finance, and mobile applications, where data cannot be centralized due to legal, ethical, or security concerns.

How Federated Learning Works?

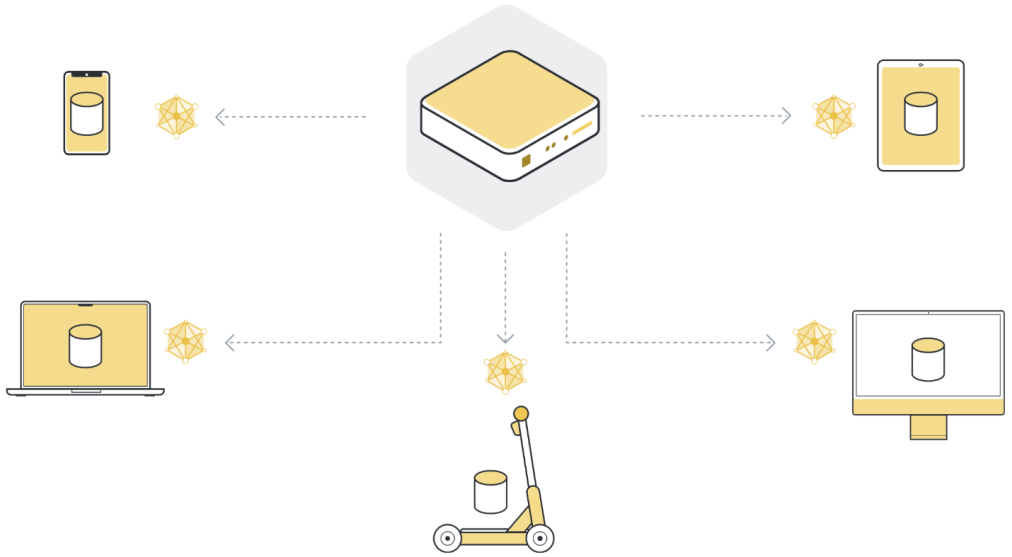
Step 1: Initialize global model

We start by initializing the model on the server. This is exactly the same in classic centralized learning: we initialize the model parameters, either randomly or from a previously saved checkpoint.



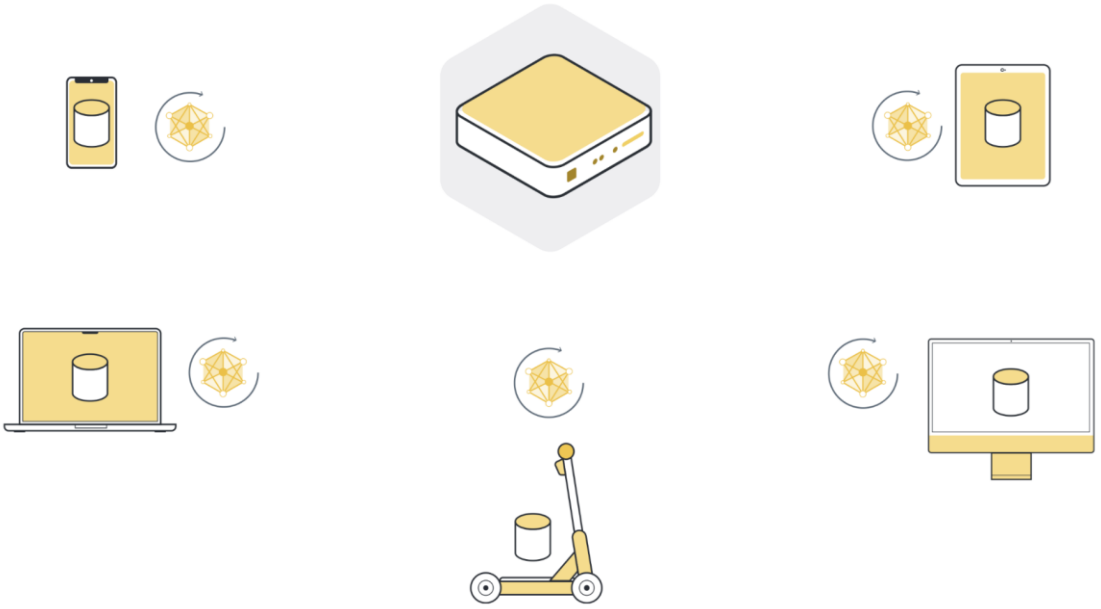
Step 2: Send model to a number of connected organizations/devices (client nodes)

Next, we send the parameters of the global model to the connected client nodes (think: edge devices like smartphones or servers belonging to organizations). This is to ensure that each participating node starts its local training using the same model parameters. We often use only a few of the connected nodes instead of all nodes. The reason for this is that selecting more and more client nodes has diminishing returns.



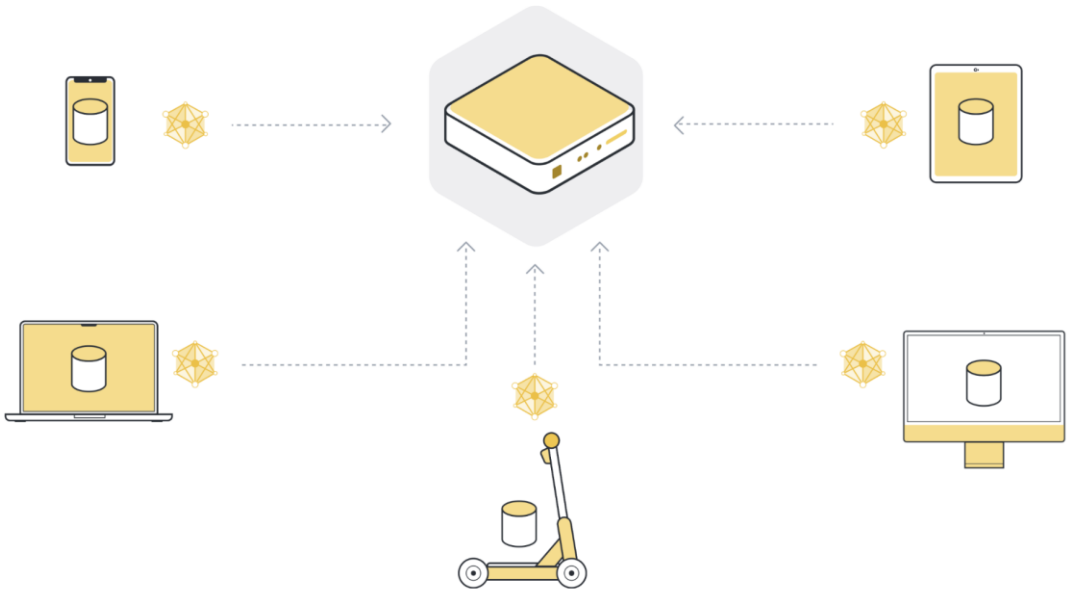
Step 3: Train model locally on the data of each organization/device (client node)

Now that all (selected) client nodes have the latest version of the global model parameters, they start the local training. They use their own local dataset to train their own local model. They don't train the model until full convergence, but they only train for a little while. This could be as little as one epoch on the local data, or even just a few steps (mini-batches).



Step 4: Return model updates back to the server

After local training, each client node has a slightly different version of the model parameters they originally received. The parameters are all different because each client node has different examples in its local dataset. The client nodes then send those model updates back to the server. The model updates they send can either be the full model parameters or just the gradients that were accumulated during local training.



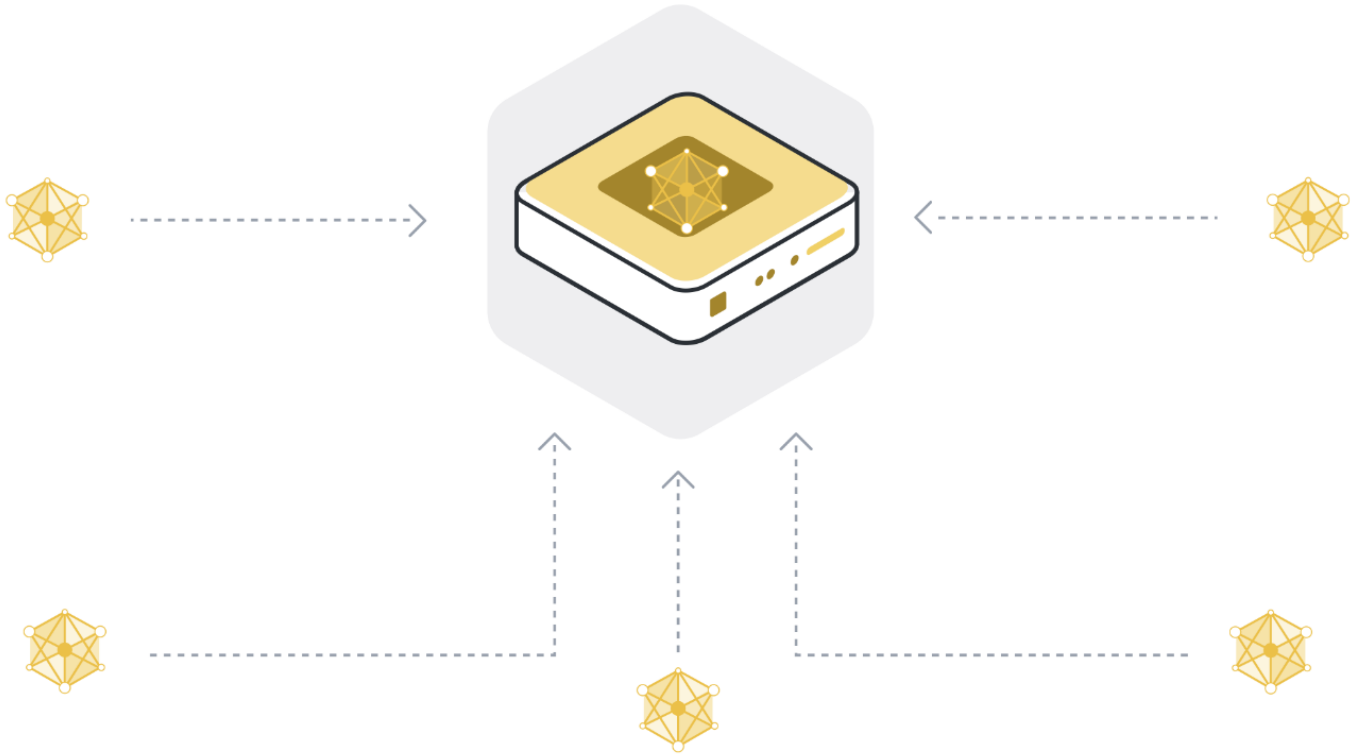
Step 5: Aggregate model updates into a new global model

The server receives model updates from the selected client nodes. If it selected 100 client nodes, it now has 100 slightly different versions of the original global model, each trained on the local data of one client. But didn't we want to have one model that contains the learnings from the data of all 100 client nodes?

In order to get one single model, we have to combine all the model updates we received from the client nodes. This process is called aggregation, and there are many different ways to do it.

The most basic way is called Federated Averaging, often abbreviated as FedAvg**.** It is one of the most widely used algorithms in the field of federated learning. It serves as the cornerstone for aggregating locally trained models from multiple clients to update the global model in a decentralized learning environment.

FedAvg takes the 100 model updates and, as the name suggests, averages them. To be more precise, it takes the weighted average of the model updates, weighted by the number of examples each client used for training. The weighting is important to make sure that each data example has the same "influence" on the resulting global model. If one client has 10 examples, and another client has 100 examples, then - without weighting - each of the 10 examples would influence the global model ten times as much as each of the 100 examples.



Step 5: Repeat steps 1 to 4 until the model converges

Steps 1 to 4 are what we call a single round of federated learning. The global model parameters get sent to the participating client nodes (step 1), the client nodes train on their local data (step 2), they send their updated models to the server (step 3), and the server then aggregates the model updates to get a new version of the global model (step 4).

During a single round, each client node that participates in that iteration only trains for a little while. This means that after the aggregation step (step 4), we have a model that has been trained on all the data of all participating client nodes, but only for a little while. We then have to repeat this training process over and over again to eventually arrive at a fully trained model that performs well across the data of all client nodes.

Flower: A Federated Learning Framework

In this project, the implementation of Federated Learning is facilitated by **Flower** (FL), an open-source framework specifically designed to build, experiment with, and deploy federated learning systems. Flower provides a flexible and extensible architecture that supports a variety of use cases, including cross-silo and cross-device federated learning. It is particularly suitable for research and production-level applications.

How Flower Works?

Server and Client Architecture:

- The server manages the global model and coordinates the aggregation of updates from the clients.
- The clients train local models on their data and communicate updates back to the server.

Strategy Customization:

- Flower provides built-in federated learning strategies such as Federated Averaging (FedAvg).
- Users can define custom strategies for model aggregation and optimization.

Communication Protocol:

- Flower uses **gRPC** for efficient communication between the server and clients. is a modern open-source framework for high-performance, cross-platform communication. It enables the execution of functions on remote servers as if they were local, using the Remote Procedure Call (RPC) paradigm.
- In federated learning, **gRPC** facilitates secure and efficient communication between the central server and distributed clients. Its lightweight, scalable nature makes it perfect for transferring model updates and instructions in real-time.

Federated Learning Architecture in This Project

In this project, we implemented a federated learning architecture using a **server** and **two clients** to simulate a distributed training environment. The training data was divided between the two clients in a specified ratio, ensuring that each client had access to a unique subset of the dataset. This division of data reflects a real-world federated learning scenario, where data resides across multiple devices or institutions and cannot be shared directly due to privacy or security constraints.

Architecture Components:

Server:

- Acts as the central coordinator for the federated learning process.
- Receives model updates (weights and metrics) from the clients after local training.
- Aggregates the updates using the Federated Averaging (FedAvg) algorithm to produce a global model.
- Distributes the updated global model back to the clients for further training in subsequent rounds.

Clients:

- Each client is responsible for local training on its respective dataset.
- Utilizes the global model sent by the server as the starting point for training.
- Sends the updated model parameters to the server after completing local training for a specified number of epochs.

Data Distribution:

The training data was divided between the two clients to simulate **non-i.i.d (non-independent and identically distributed)** data conditions. This means that the data distribution on each client may not be uniform, further emphasizing the importance of federated learning to address such imbalanced and distributed scenarios.

Federated Learning Workflow:

Initialization:

The server initializes the global model with **random weights** and sends it to all clients.Each client receives the model and prepares for local training.

Local Training:

Clients train the model locally using their respective datasets.Each client computes the updated weights and optionally evaluates the model on its validation data.

Aggregation:

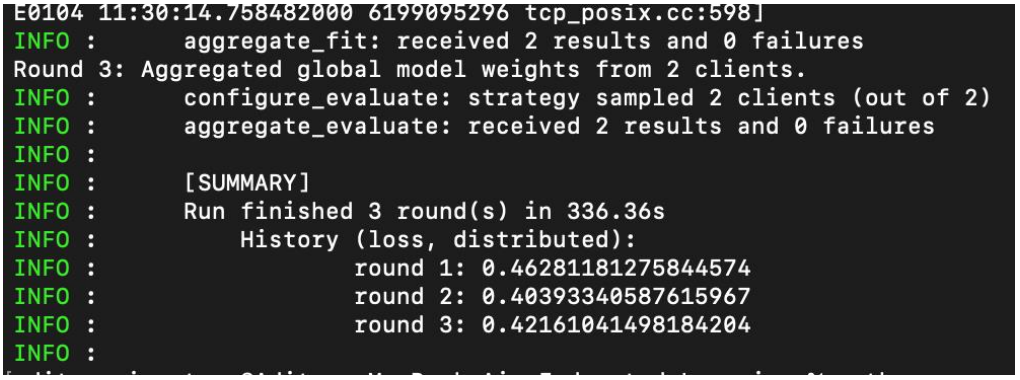
Clients send their locally trained model parameters to the server.The server aggregates the updates using the FedAvg algorithm to produce a new global model.

Iterations:

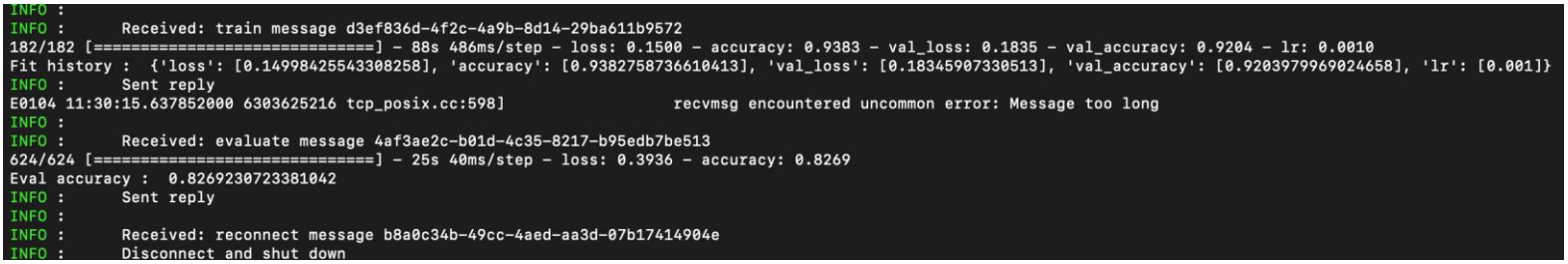
The updated global model is sent back to the clients.The process repeats for a predefined number of rounds, progressively improving the model's performance. model that has been trained on all the data of all participating client nodes, but only for a little while. We then have to repeat this training process over and over again to eventually arrive at a fully trained model that performs well across the data of all client nodes.

Server and Client in CLI

Server CLI Snapshot at the end of Federated Learning



Client CLI Snapshot at the end of Federated Learning



Conclusion

This project successfully demonstrated the use of federated learning for pneumonia detection using chest X-ray images. By leveraging machine learning, deep learning, and transfer learning techniques, we developed a robust predictive model. Incorporating federated learning with the Flower framework allowed us to train the model in a distributed manner while maintaining data privacy. The use of Federated Averaging (FedAvg) effectively aggregated local updates, resulting in a globally optimized model. This approach highlights the potential of federated learning to enable collaborative AI solutions in healthcare, paving the way for privacy-preserving, scalable, and efficient diagnostic tools.

References

1. **Flower Ai**
   https://flower.ai/docs/framework/tutorial-series-get-started-with-flower-pytorch.html#
2. **Federated Learning**
   - https://www.ibm.com/docs/it/watsonx/saas?topic=models-federated-learning
   - https://moodle2.unime.it/pluginfile.php/28427347/mod_resource/content/0/14-federated-learning.pdf
3. **CNN**
   - https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/