UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

JOÃO HENRIQUE ARAÚJO ROCHA
Orientador: Prof. Dr. Rodrigo Cezar Pedrosa
Coorientador: Prof. M. Sc. Lauro Angelo Gonçalves de Moraes

APRENDIZADO FEDERADO E O IMPACTO DE CLIENTES NA REDE

Ouro Preto, MG
2022

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

JOÃO HENRIQUE ARAÚJO ROCHA

APRENDIZADO FEDERADO E O IMPACTO DE CLIENTES NA REDE

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Rodrigo Cezar Pedrosa
Coorientador: Prof. M. Sc. Lauro Angelo Gonçalves de Moraes

Ouro Preto, MG
2022

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

João Henrique Araújo Rocha

Aprendizado Federado e o impacto de clientes na rede

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 03 de Novembro de 2022.

Membros da banca

Rodrigo César Pedrosa Silva (Orientador) - Doutor - Universidade Federal de Ouro Preto
Lauro Angelo Goncalves de Moraes (Coorientador) - Mestre - Programa de Pós-Graduação em Ciência da Computação (PPGCC-UFOP)
Pedro Henrique Lopes Silva (Examinador) - Doutor - Universidade Federal de Ouro Preto
Guilherme Augusto Lopes Silva (Examinador) - Bacharel - Universidade Federal de Ouro Preto

Rodrigo César Pedrosa Silva, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 03/11/2022.

Documento assinado eletronicamente por **Rodrigo Cesar Pedrosa Silva**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 02/12/2022, às 12:54, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0421006** e o código CRC **731CC7A8**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Dedico este trabalho a minha família, minha esposa Mariana e a minha filha Clarice

Agradecimentos

Agradeço a Universidade Federal de Ouro Preto pela qualidade de ensino e atenção com os dicentes, o departamento de computação, DECOM, pelo cuidado com o curso de alta qualidade, todos os professores do DECOM, em especial Prof. Dr. Joubert de Castro Lima por todas lições que recebi, pela paciência comigo e pelas oportunidades que me apresentou durante todo o curso. Aos meus orientadores Prof. Dr. Rodrigo Cezar Pedrosa e Prof. M. Sc. Lauro Angelo Gonçalves de Moraes. Agradeço também aos meus pais e familiares que sempre me apoiaram em qualquer decisão tomada durante todo o percurso, o laboratório HPC no qual fiz parte e aos amigos que fiz lá dentro e vou carregar para toda vida. Por fim agradeço a todos integrantes da Koxapa por todos os grupos de estudo, apoio e risadas durante minha graduação.

Resumo

Este trabalho apresenta um estudo de como o aprendizado federado se comporta quando se utilizam amostragem de tamanhos diferentes ao realizar o treinamento distribuído de um modelo de redes neurais profundas (DNN) usando a técnica de agregação dos pesos "Federated averaging"(FedAvg). Utilizamos o modelo de rede neural profunda *MobileNetV2* presente no framework *Tensorflow* 2.x. O treinamento federado foi implementado com o framework *Flower* 1.0 utilizando a base de dados de imagens CIFAR10. Os resultados apresentam que em um pequeno conjunto de clientes a variação de amostra de clientes nos passos de treinamento e predição impacta mais no tempo de processamento e em alguns casos, na qualidade do modelo. Entretanto, o caso base executado localmente apresentou melhores resultados em todas as métricas utilizadas por conta da quantidade de dados disponível em comparação com a configuração do aprendizado federado.

Palavras-chave: Aprendizado de Máquina. Aprendizado Federado. Tensorflow. Keras. Redes Neurais Profundas

Abstract

This work presents a study of how federated learning behaves when using different sample sizes when performing the distributed training of a model of deep neural networks (deep learning) using the "Federated averaging" (Fed Avg) algorithm. We use the MobileNetV2 deep learning model present in the TensorFlow 2.x framework. The federated training was implemented with the Flower 1.0 framework using the CIFAR10 image database. The results of training and tests were collected and analyzed. Experiments shows that in a small set of clients, the variation in the sample of clients in the training and prediction steps impacts more on the processing time and, in some cases, on the quality of the model. But the baseline was trained locally and presented the best results in all three metrics observed because of the amount of data availiable in comparison with the Federated Learning setup.

Keywords: Machine Learning, Federated Learning, TensorFlow, Keras, Deep Neural Network

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	Application Programming Interface (Interface de programação de aplicação)
ASIC	Application-Specific integrated circuit (Circuito integrado de aplicação específica)
CNN	Convolutional Neural Network (Rede Neural Convolucional)
CPU	Central Processing Unit (Unidade de Processamento Central)
DECOM	Departamento de Computação
GPU	Graphic processing Unit (Unidades de processamento gráfico)
ML	Machine Learning (Aprendizado de Máquina)
SGD	Stochastic Gradient Descent (Descida Gradiente Estocástica)
TPU	Tensor processing Unit (Unidade de processamento de Tensor)
UFOP	Universidade Federal de Ouro Preto

Sumário

1 Introdução

Nos últimos anos, tem o número cresce pesquisas relacionadas ao aprendizado federado (FL) (NADA, a)(MCMAHAN et al., 2016). Um dos precursores é a necessidade de distanciar dos métodos tradicionais de inteligência Artificial (IA) que precisam dos dados centralizados. O crescente número de dados, o aumento preocupação com a privacidade e crescente número de dispositivos disponíveis tem fomentado esse distanciamento e a busca por métodos distribuídos e descentralizados aumentado (BEN-NUN; HOEFLER, 2018). Em (NGUYEN et al., 2021) é descrito a necessidade do FL para auxiliar a medicina onde os dados dos pacientes são sensíveis, mas o resultado dos exames e os laudos médicos são essenciais para o treinamento de modelos de IA podendo ser utilizado para a generalização de problemas na medicina (LIU et al., 2021).

O FL é uma forma de aprendizado de máquina descentralizado para realizar o processo de treinamento dos pesos de maneira centralizada, que trata os clientes sem conhecer informações do mesmo. Ele utiliza do treinamento distribuído de modelos.

O treinamento distribuído é uma forma de treinar modelos de ML quando o cientista de dados quer avaliar grandes modelos, ou obter resultados parecidos com menos tempo de processamento. Existem três principais padrões utilizados para distribuir um modelo de ML: distribuição do modelo, distribuição dos dados e distribuição do modelo e dos dados. (BEN-NUN; HOEFLER, 2018).

A distribuição dos modelos é quando é possível compartilhar neurônios de uma rede em várias máquinas ou cores paralelamente. Também seria possível executar cada camada de um modelo em diferentes máquinas, no caso de uma rede neural profunda ser utilizada.

(SMITH et al., 2017) compara algoritmos centralizados com algoritmos descentralizados. O algoritmo centralizado utiliza de um servidor central, que conecta todos os clientes. Já no caso do descentralizado, os clientes podem comunicar-se entre si. Ele mostrou ser possível que algoritmos descentralizados, ao obter convergência parecida e sob determinada circunstâncias, tenham uma melhoria significativa, em casos de alto tráfego de informações.

Um dos desafios dos modelos distribuídos e descentralizados é a comunicação. Ela deixa de ser local e passa por uma rede de dispositivos que podem estar com uma latência alta, desconectados, desligados ou defeituosos (KAIROUZ et al., 2019).

O aprendizado federado possui alta heterogeneidade, em termos de hardware e dados disponíveis nos clientes (KAIROUZ et al., 2019). Essas características podem ser úteis quando se trata de privacidade, mas não perfeitas, pois ao se ter acesso a várias bases de dados de diferentes clientes, há um risco ao anonimato destes dados. O trade-off é ter que trafegar parâmetros pela rede, estando sujeito a problemas de latência e disponibilidade dos clientes (MCMAHAN et al.,

2016).

1.1 Justificativa

A motivação dos autores é explorar o potencial do aprendizado federado e entender qual seria um bom trade-off entre treinamento x acurácia dos modelos gerados, em relação ao número de clientes presentes na rede. Como exemplificado Seção 1, quanto mais dados, seja eles imagens, áudios, laudos, maior a chance de se obter um bom modelo de IA. Isso poderia justificar e motivar adesão de outros hospitais sem que os dados sensíveis sejam expostos.

1.2 Objetivos

Entender o impacto nos tempos de treinamentos e nas qualidades do modelo gerado ao se executar e avaliar diferentes configurações de treinamento federado para o modelo *MobileNetV2* na tarefa de classificação de imagens usando a base de dados CIFAR10.

1.2.1 Objetivos Específicos

Temos os seguintes objetivos específicos:

- Variar o número de máquinas
- Variar o número de instâncias por máquina
- Variar o número de amostra por instâncias
- Verificar o efeito destes fatores na qualidade do modelo e no tempo de treinamento.

1.3 Metodologia

Para desenvolvimento dos testes foi utilizado do Python na versão 3.8 hospedado na nuvem da *Google Cloud Plataform* (GCP). Foi utilizado a base de dados Cifar10 disponibilizada pelo framework Keras e o modelo *MobileNetV2* também disponível no Keras. O modelo foi treinado utilizando o framework *Flower* com a estratégia de agregação dos parâmetros dos modelos treinados de *FedAvg*. Utilizamos também o modelo de treinamento de aprendizado federado, limitado a no máximo quatro clientes por máquinas. A partir do resultado obtido utilizou de ferramentas disponíveis do Scikit-learn. Os resultados foram comparados através gráficos e tabelas elaboradas pelo autor.

1.4 Organização do Trabalho

O restante do trabalho se organiza da seguinte forma:

Capítulo 1: Introdução.

Capítulo 2: Redes neurais e Aprendizado Federado.

Capítulo 3: Desenvolvimento.

Capítulo 4: Resultados.

Capítulo 5: Conclusão.

A Seção 2 consiste nos detalhes da revisão bibliográfica e conceitos básicos para facilitar o entendimento deste trabalho. A Seção 3 descreve como foi implementado o algoritmo de teste. A seção 4 apresenta os experimentos e resultados obtidos do algoritmo de teste. E a Seção 5 conclui o trabalho, e direciona os trabalhos futuros.

2 Redes neurais e Aprendizado Federado

Este capítulo apresenta alguns trabalhos da literatura sobre o impacto do Aprendizado Federado. Na primeira parte, também apresentamos alguns conceitos básicos, para facilitar o entendimento do leitor, assim como alguns conceitos utilizados apenas em nossos testes, mas pertinentes ao assunto.

2.1 Rede Neurais Profundas

Redes neurais profundas são camadas de componentes conectados, simulando neurônios de um cérebro biológico. Cada componente toma uma decisão individualmente, com base no input fornecido, da intensidade do sinal que iria propagar. Essas camadas consistem em neurônios, sinapses, pesos e funções (GAVRILOVA,). Essas redes podem aparecer por vários motivos, como descobrir um tipo específico de raça de cachorro ou reconhecer números em imagens. A Figura 2.1 representa uma versão simplificada de uma rede neural. Em verde esta representando a camada de input, em azul as camadas escondidas e em amarelo a camada de output.

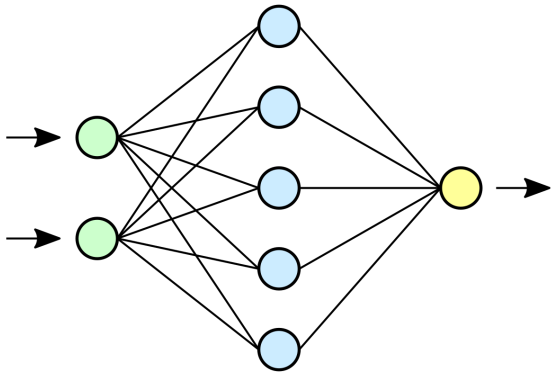

Figura 2.1 – Representação simplificada de uma Rede Neural

O treinamento de uma rede neural para classificação de imagens consistem em obter amostras da base de dados, conhecidas como lotes (*batch* em inglês), utilizar o modelo para calcular os pesos (*weigths* em inglês) e vieses (*bias* em inglês) com base nessas amostras por várias epocas (*epoch* em inglês). Isso resulta em um modelo que identificou padrões na base de dados fornecida como input e consegue gerar outputs, neste caso, classificar uma imagem.

O FL basicamente inclui mais um passo no processo de treinamento. O algoritmo 1 mostra os passos para o aprendizado federado. É definido um número de iterações globais em I e uma quantidade total de clientes conectados ao servidor definido por K. Os pesos globais iniciais podem ou não ser inicializados em g. O algoritmo começa para cada cliente k conectado atualizando o peso local com o peso global g. Cada cliente utiliza a própria base de dados para

treinar o modelo em $f(w)$ gerando novos pesos da rede em W. Após todos os clientes terminarem, uma função G é aplicada a todos os pesos dos clientes W_k para agregar os resultados no servidor. Isso conclui uma iteração do FL que se repete até terminar em I.

Algoritmo 2.1: Aprendizado Federado

```
I ← Iteracoes;
K ← clientes;
g ← W_0 ;
while I >= 0 do
    for k ∈ K do
        k ← g;
        W ← f(w);
    g ← G(W_k);
    I ← I − 1;
```

2.2 Descida Gradiente Estocastica

Também conhecido como SGD (Stochastic Gradient Descent), a Descida Gradiente Estocástica é um algorítmo iterativo de otimização que busca o ponto mínimo de uma função utilizando o gradiente da mesma (RUDER, 2016).

O cálculo do gradiente é feito por uma estimativa que considera uma amostra aleatória da base de dados chamada de *minibatch*. É necessária essa amostra, pois custoso calcular com base em todo conjunto de treinamento.

A Figura 2.2 representa de maneira intuitiva o SGD como uma partícula descendo em direção ao menor ponto de um espaço curvo.

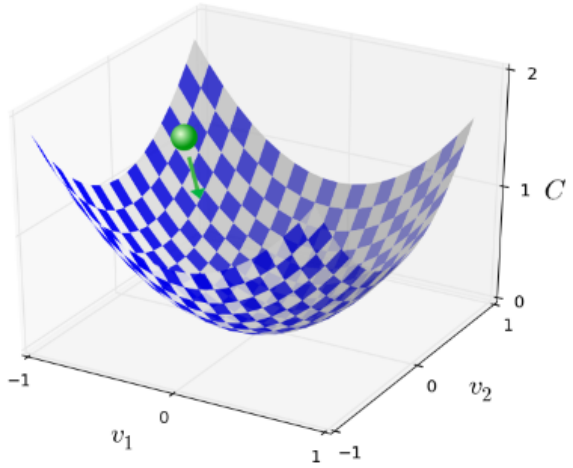

Figura 2.2 – Representação gráfica do SGD

Fonte: (NIELSEN, 2016)

2.3 Backpropagation

O algoritmo *backpropagation* é comumente utilizado em redes neurais para propagar correções na rede, visando melhorar em neurônio o peso e o bias vinculado a ele. Repetindo esse processo para todas as camadas da rede dá a ideia de propagação retroativa. (NADA, b).

Ele pode ser dividido em quatro fases principais segundo (ROJAS, 2013).Primeiro ocorre *feed-forward computation* onde os valores das camadas ocultas são utilizados para computar a camada de output. Depois é computado o erro do output da rede com o valor desejado, isso é a fase de *backpropagation* na camada de output.

Após a determinação do erro na camada de output é calculado o erro para as camadas ocultas na terceira fase sendo chamada *backpropagation* da camada oculta. Por fim, o último passo é atualizar os pesos depois que todos os pesos forem ajustados de erro em toda rede.

A Figura 2.3 mostra de maneira simplificada a propagação dos pesos na rede. O output é comparado com o resultado desejado e calcula quais nos tem maior peso no resultado correto. Esse valor é corrigido em cadeia para cada nó das camadas anteriores até o início da rede.

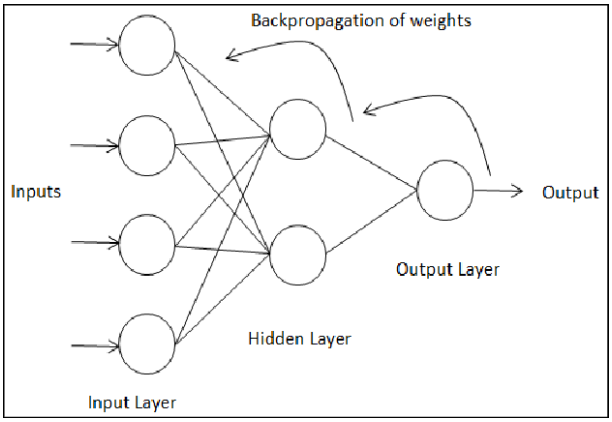

Figura 2.3 – Representação simplificada do backpropagation

2.4 Keras

Keras é uma biblioteca de código aberto, que dispõe uma API em Python simples, flexível e consistente para desenvolvimento de trabalhos com deep learning. Keras age como uma interface para o *tensorflow*. (KERAS, 2021)

Tensorflow é um framework de ML. Contém uma vasta comunidade e foi originalmente criado pelos engenheiros do *tensorflow*Google Brain Team, que funciona em larga escala e em ambientes heterogêneos.

(ABADI et al., 2016)

Eles fornecem diversas ferramentas para a comunidade de ML. Entre elas estão modelos, camadas para os modelos, recursos para tolerância a falhas, otimizadores e muitas outras. Neste

trabalho utilizamos um modelo pré-definido e um otimizador para o mesmo.

2.5 Flower

Flower é um framework de FL, desenvolvido pela Ele. Os quatro princípios seguidos pelo *Flower* são: customizável, extensível, legibilidade e independência. É customizável por permitir uma grande variedade de configurações. Extensível por permitir que os componentes sejam sobrescritos por novos algoritmos, com o estado da arte, legível pois foi construído visando ser fácil, de dar manutenção Por fim, é agnóstico em relação aos *frameworks* de ML, permitindo ao usuário utilizar os pontos fortes de cada um deles. (BEUTEL et al., 2020)

2.6 Trabalhos Relacionados

O primeiro trabalho relacionado ao FL é (MCMAHAN et al., 2016) que mostra um método prático de média (*FedAvg*) e avalia desempenho dos experimentos. Em comparação com cinco modelos de ML e quatro base de dados observaram ser possível reduzir a quantidade de troca de informações nas iterações do FL de 10 a 100 vezes menos do que comparado ao método SGD síncrono.

Na Equação (2.1) K é número de clientes, n o tamanho da base de dados, n_k o tamanho da base de dados do cliente e $f_k(w)$ a função de perda do cliente.

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} f_k(w) \tag{2.1}$$

Foi proposto também em (KARIMIREDDY et al., 2019) a redução ainda maior da comunicação apresentando um novo algoritmo chamado *SCAFFOLD* que corrige a deriva de conceito dos modelos localmente. Os resultados mostraram que o algoritmo proposto consegue obter resultados melhores com menos iterações do que o inicialmente proposto *FedAvg* e o tradicional SGD.

Como mencionado na Seção 2.2, o algoritmo SGD utiliza de amostras aleatórias para tentar calcular uma aproximação do gradiente. O trabalho (KESKAR et al., 2016) explora o problema inerente ao utilizar grandes amostras a cada epoca que é reduzir a qualidade do modelo em termos de generalização, apresentando estratégias que ajudam a eliminar essa diferença de qualidade em reconhecimento de imagens.

O estado da arte em base de dados heterogêneas segundo (NADA, c) é uma combinação de algoritmos chamados de *SiloBN*, *ASAM* e *SAM* descrito por (CALDAROLA; CAPUTO; CICCONE, 2022). Eles correlacionam que uma má generalização do modelo está relacionado a “pontas” nos gráficos. Outro problema seria que caso ocorresse um aumento das dimensões

do valor de perda, as “pontas” poderiam ser consideradas pontos “planos” que teoricamente se correlacionam com uma boa generalização.

Pensando nisso, eles propuseram os algoritmos *ASAM* e *SAM* são utilizados no lado dos clientes no FL visando procurar no espaço de busca da função de perda uma região mais "plana"ao mesmo tempo, adaptam as pontas conforme as dimensões. Por fim, o algoritmo *SiloBN* é utilizado no lado do servidor proposto por (ANDREUX et al., 2020) para melhorar a confiabilidade de base de dados heterogêneas na classificação de imagens de tumores.

Os resultados mostraram que essa combinação é efetiva entre várias configurações, sendo a melhor a combinação citada. Para provar a generalização, foi construído bases de dados corrompidas do CIFAR10 descrito na Seção 3.3 com até cinco níveis de severidade. Os resultados mostraram que mesmo no pior cenário, o algoritmo ainda se mostra eficaz.

3 Desenvolvimento

Nesta seção, detalhamos como foram construídos os algoritmos utilizados e a arquitetura do mesmo. As definições de parâmetros vão ser discutidas em 4, posteriormente.

3.1 Métricas

Destacamos 3 métricas no processo de avaliação dos resultados. Elas são a F1, Acurácia e Perda. Antes de definir o que são essas métricas, precisamos definir quatro conceitos: Falso positivo (FP), Falso negativo(FN), positivo verdadeiro(TN) e Positivo falso(TP).

FP é quando o modelo afirma ter uma classe no input, quando na verdade essa classe não está presente. Isso é como se o modelo informasse que existe um caminhão na imagem, quando, na verdade não existe um.

FN é quando o modelo prediz que uma classe não está presente no input, quando, na verdade essa classe estava presente. Seria classificar que não existe um caminhão em uma imagem que tem um caminhão.

TN é quando o modelo prediz de maneira correta que no input não possui determinada classe. Na nossa base de dados, ele classificaria uma imagem sem caminhão como uma imagem sem caminhão.

TP é quando o modelo prediz de maneira correta a classe que o input representa. Na nossa base de dados, isso seria classificar um caminhão como de fato um caminhão.

A acurácia é a métrica utilizada para dizer o número de predições corretas feitas pelo modelo, em comparação com todas as predições feitas como mostra na equação (3.1).

$$Acuracia = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

A métrica F1Score é uma média harmônica da precisão e da sensibilidade do modelo. Precisão é calculada com base na quantidade de predições feitas corretamente. A Sensibilidade diz sobre quantos casos positivos o modelo classificou corretamente em comparação com todos os casos positivos da base de dados.

As equações (3.2), (3.3) e (3.4) representam respectivamente os cálculos da Sensibilidade, Precisão e F1Score.

$$Sensibilidade = \frac{Positivosverdadeiros}{Positivosverdadeiro + FalsePositivo} \tag{3.2}$$

$$Preciso = \frac{PositivosVerdadeiros}{Positivoverdadeiro + FalsoPositivo} \tag{3.3}$$

$$F1Score = 2 * \frac{Preciso * Sensibilidade}{Preciso + Sensibilidade} \tag{3.4}$$

A ideia do F1Score é ter em uma única métrica para ponderar as duas razões (Precisão e Sensibilidade) de uma maneira balanceada de forma que qualquer alteração drástica em ambos reflete no resultado. (NADA, d)

3.2 Arquitetura

A arquitetura utilizada é composta de 7 máquinas virtuais, disponíveis no *Google Cloud Platform*. Também conhecido como GCP, ele é uma plataforma em que é possível utilizar de vários recursos de hardware e software para realizar experimentos, ou até mesmo implantar soluções na nuvem. As máquinas se dividem em 6 clientes e 1 servidor, com quatro cores e 16 gigas de memória RAM. A imagem 3.1 representa as conexões que cada cliente tem com o servidor. Cada um deles estabelece conexão com 4 serviços iguais, mas em processos diferentes, executando o mesmo código que será discutido na Seção 3.6.

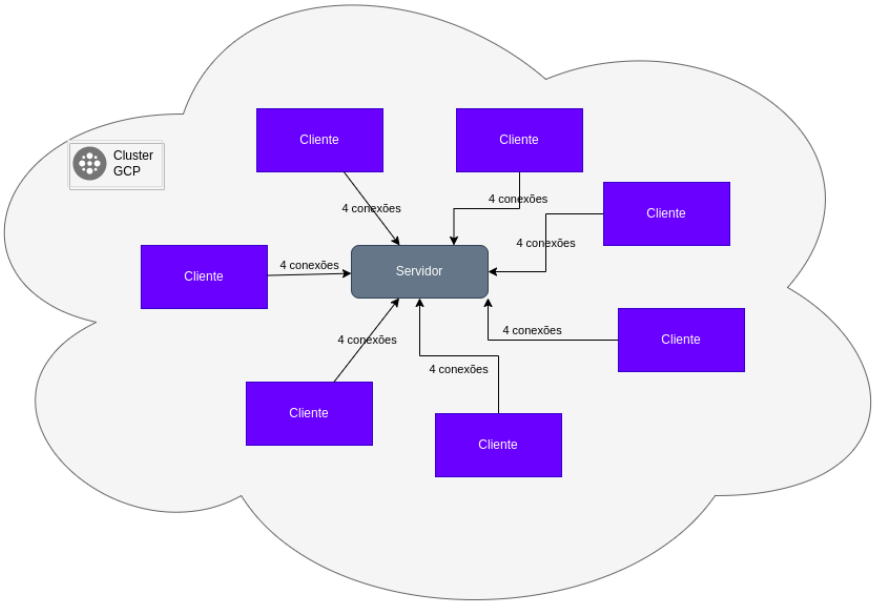

Figura 3.1 – Representação da arquitetura utilizada nos testes. Em roxo os clientes e cinza-escuro o servidor

Fonte: Elaborado pelo autor

Foi estabelecido que cada cliente teria no máximo 4 serviços para se igualar ao número de cores.

3.3 Base de Dados

Foi decidido utilizar a base de dados CIFAR10 que possui 60.000 imagens coloridas com 32x32 pixels. Existem 6000 imagens por classe, e são 10 classes no total sendo elas avião, automóvel, pássaro, gato, cervo, cachorro, sapo, cavalo, barco e caminhão. Das 60.000 imagens, 50.000 são de treino e 10.000 de teste. A Figura 3.2 exemplifica a classe barco da base de dados.

Figura 3.2 – Exemplos das 10 classes presente na base de dados

Fonte: Elaborado pelo autor

As imagens são divididas em seis lotes disjuntos de 10.000, sendo 5 lotes de treino e 1 de teste. O lote de teste possui 1000 imagens aleatórias de cada classe. Os lotes de treino possuem as imagens restantes em ordens diferentes, mas alguns lotes podem possuir mais imagens de uma classe do que outros. No total, existem exatamente 5000 imagens para cada classe. As classes são todas mutuamente exclusivas (KRIZHEVSKY, 2009). Na Figura 3.3 podemos observar alguns exemplos de cada classe presente na base de dados.

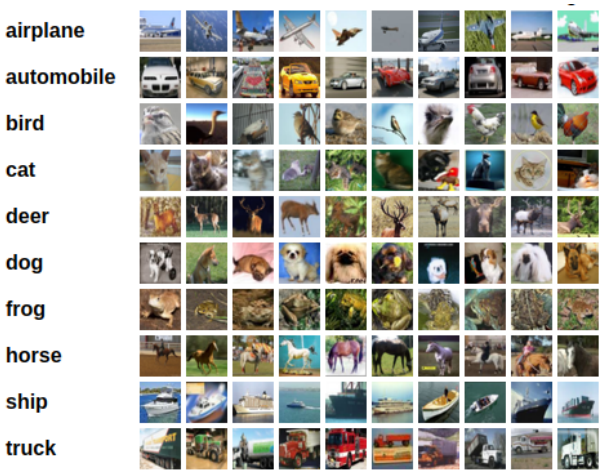

Figura 3.3 – Exemplos das 10 classes presente na base de dados

Fonte: https://www.cs.toronto.edu/ kriz/cifar.html acesso 20/10/2022

3.4 Modelo

O modelo utilizado é o MobileNetV2 . Ele é uma rede neural profunda convolucional que tem o objetivo de obter bons resultados em dispositivos móveis. Com 3.5 milhões de parâmetros e 105 camadas de profundidade (SANDLER et al., 2018). Ele foi escolhido por ser o modelo disponível na API do keras com o menor número de parâmetros.

3.5 Servidor

O servidor foi construído utilizando o *framework Flower*. Dessa forma, só foi preciso definir os modelos, uma estratégia de agregação e uma função de avaliação. É possível também variar a amostragem de serviços usados nos passos de treino e no passo de avaliação.

No trecho de código 3.1 se encontram todas as funções necessárias para o funcionamento do servidor.

Listing 3.1 – Imports necessários para o servidor

```
1 import time
2 from typing import Dict, Optional, Tuple
3 from pathlib import Path
4 import argparse
5
6 import flwr as fl
7 import tensorflow as tf
8 from keras import backend as K
9 from keras import models, layers
```

Conforme as equações definidas em 3.1, o trecho 3.2 define o cálculo de Sensibilidade na linha 1, precisão na linha 8 e F1Score na linha 15.

Listing 3.2 – Implementação da métrica F1Score

```
1  def recall_m(y_true, y_pred):
2      true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0,
           1)))
3      possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
4      recall = true_positives / (possible_positives + K.epsilon()
           )
5      return recall
6
7
8  def precision_m(y_true, y_pred):
9      true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0,
           1)))
10     predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
11     precision = true_positives / (predicted_positives + K.
           epsilon())
12     return precision
13
14
```

```
15 def f1_m(y_true, y_pred):
16     precision = precision_m(y_true, y_pred)
17     recall = recall_m(y_true, y_pred)
18     return 2 * ((precision * recall) / (precision + recall + K.
           epsilon()))
```

O modelo definido é o modelo padrão do Keras para o *MobileNetV2* sem o uso dos pesos como mostra o trecho 3.3 nas linhas 1 e 11. As linhas 2 a 10 são apenas definições de variáveis para facilitar a escrita dos roteiros de teste.

Listing 3.3 – definição e compilação do modelo

```
1     model = tf.keras.applications.MobileNetV2(input_shape=(32,
          32, 3), weights=None, classes=10)
2
3     parser = argparse.ArgumentParser()
4     parser.add_argument("--fraction_fit", default=1.0, type=
          float)
5     parser.add_argument("--fraction_evaluate", default=1.0,
          type=float)
6     parser.add_argument("--min_fit_clients", default=2, type=
          int)
7     parser.add_argument("--min_evaluate_clients", default=2,
          type=int)
8     parser.add_argument("--min_available_clients", default=2,
          type=int)
9     args = parser.parse_args()
10    run = f'{args.fraction_fit}_{args.fraction_evaluate}_{args.
          min_fit_clients}_{args.min_evaluate_clients}_{args.
          min_available_clients}'
11    model.compile("adam", "sparse_categorical_crossentropy",
          metrics=["accuracy", f1_m])
```

Neste último trecho, 3.4 na linha 1 há o início da definição da estratégia utilizada. Os argumentos estão definidos da linha 2 até a linha 10.

- fraction_fit = porcentagem de clientes utilizados no passo de treinamento
- fraction_evaluate = porcentagem de clientes utilizados no passo de avaliação
- min_fit_clients = quantidade mínima de clientes para o início do treino
- min_evaluate_clients = quantidade mínima de clientes para o início da avaliação

- min_available_clients = quantidade mínima de clientes para o início do processo de aprendizado federado

Listing 3.4 – definicao da estratégia e servidor

```
1   strategy = fl.server.strategy.FedAvg(
2       fraction_fit=args.fraction_fit,  # 0.3
3       fraction_evaluate=args.fraction_evaluate,  # 0.2
4       min_fit_clients=args.min_fit_clients,
5       min_evaluate_clients=args.min_evaluate_clients,
6       min_available_clients=args.min_available_clients,
7       evaluate_fn=get_evaluate_fn(model, args),
8       on_fit_config_fn=fit_config,
9       on_evaluate_config_fn=evaluate_config,
10      initial_parameters=fl.common.ndarrays_to_parameters(
            model.get_weights()),
11  )
12
13  fl.server.start_server(
14      server_address="[::]:8080",
15      config=fl.server.ServerConfig(num_rounds=25),
16      strategy=strategy,
17  )
```

Os próximos quatro parâmetros são os evaluate_fn, on_fit_config, on_evaluate_config e initial_parameters e estão representados no trecho 3.5. Respectivamente eles representam:

- Uma função que define a forma que o servidor processará o passo de avaliação. No nosso caso, como dito anteriormente, foram definidos que 5000 imagens seriam utilizadas para a validação como mostra na linha 6.
- Uma função a ser executada antes de cada passo de treinamento nos clientes. Definida na linha 23, ela recebe a interação do servidor como parâmetro e retorna um objeto arbitrário. Esse objeto é usado pelo cliente da maneira que o usuário quiser. Definimos que o tamanho do lote será 32, e que o número de epocas que cada cliente executará é 1 na primeira rodada do servidor e 2 nas seguintes.
- Uma função a ser executada antes de cada passo de avaliação dos clientes. Ela é definida na linha 31, e define a quantidade de passos de validação que serão executados nos clientes. Foi definido que serão 5 nos primeiros 3 rounds e 10 nos restantes.
- Parâmetros iniciais do modelo no servidor. Esses parâmetros foram iniciados arbitrariamente, com os dados do próprio modelo não treinado na linha 10 do trecho 3.4.

Listing 3.5 – funcoes auxiliares

```
def get_evaluate_fn(model, args):
    """Return an evaluation function for server-side evaluation."""

    (x_train, y_train), _ = tf.keras.datasets.cifar10.load_data()

    x_val, y_val = x_train[45000:50000], y_train[45000:50000]

    def evaluate(
            server_round: int,
            parameters: fl.common.NDArrays,
            config: Dict[str, fl.common.Scalar],
    ) -> Optional[Tuple[float, Dict[str, fl.common.Scalar]]]:
        model.set_weights(parameters)
        loss, accuracy, f1 = model.evaluate(x_val, y_val, callbacks=[
            tf.keras.callbacks.TensorBoard(
                log_dir=f'logs/server_{args.fraction_fit}_{args.fraction_evaluate}_{args.min_fit_clients}_{args.min_evaluate_clients}_{args.min_available_clients}',
                histogram_freq=1)])
        return loss, {"accuracy": float(accuracy), "loss": float(loss), "f1": f1}

    return evaluate


def fit_config(server_round: int):
    config = {
        "batch_size": 32,
        "local_epochs": 1 if server_round < 2 else 2,
    }
    return config


def evaluate_config(server_round: int):
```

```
32      val_steps = 5 if server_round < 4 else 10
33      return {"val_steps": val_steps}
```

3.6 Cliente

O Cliente assim como o Servidor foi construído utilizando o *framework Flower*. Foi necessário definir uma função de treinamento e uma função de avaliação dentro do cliente. Utilizamos o mesmo modelo, *MobileNetV2*. Para dividir os dados, fracionamos parte da base de dados de acordo com um índice de 0 a 9 aleatório ao cliente.

No trecho 3.6 da linha 1 a 11 representa todas as funções utilizadas.

Listing 3.6 – Definição dos parametros

```
1  import argparse
2  import os
3  from pathlib import Path
4
5  import tensorflow as tf
6
7  import flwr as fl
8  from keras import backend as K
9
10 from keras import layers
11 from tensorflow.python.keras import models
```

No trecho 3.7 apresenta a definição da implementação do Cliente disponibilizado pelo *framework Flower*. Nas linhas 2 a 6 definimos o construtor que armazena os dados utilizados no treinamento e nas predições, o modelo utilizado e o índice atribuído ao cliente.

O trecho da linha 8 até a 30 define a função invocada a cada iteração do aprendizado federado. Ela é responsável por atualizar o modelo, treinar localmente e retornar os resultados ao servidor principal.

Por fim, a linha 32 até 39 é definido a função de validação. Ela precisa atualizar o modelo local com os novos pesos calculados pelo servidor após o passo de agregação dos resultados. Posterior a isso ela retorna ao servidor o resultado da predição.

Listing 3.7 – Definição do client

```
1  class CifarClient(fl.client.NumPyClient):
2      def __init__(self, model, x_train, y_train, x_test, y_test,
           id):
3          self.model = model
```

```
4           self.x_train, self.y_train = x_train, y_train
5           self.x_test, self.y_test = x_test, y_test
6           self.cli_id = id
7
8       def fit(self, parameters, config):
9           self.model.set_weights(parameters)
10
11          batch_size: int = config["batch_size"]
12          epochs: int = config["local_epochs"]
13
14          history = self.model.fit(
15              self.x_train,
16              self.y_train,
17              batch_size,
18              epochs,
19              validation_split=0.1,
20          )
21
22          parameters_prime = self.model.get_weights()
23          num_examples_train = len(self.x_train)
24          results = {
25              "loss": history.history["loss"][0],
26              "accuracy": history.history["accuracy"][0],
27              "val_loss": history.history["val_loss"][0],
28              "val_accuracy": history.history["val_accuracy"][0],
29          }
30          return parameters_prime, num_examples_train, results
31
32      def evaluate(self, parameters, config):
33          self.model.set_weights(parameters)
34
35          steps: int = config["val_steps"]
36
37          loss, accuracy, f1 = self.model.evaluate(self.x_test,
                self.y_test, 32, steps=steps)
38          num_examples_test = len(self.x_test)
39          return loss, num_examples_test, {"accuracy": float(
                accuracy), "loss": float(loss), "f1": float(f1)}
```

A função mencionada para a divisão da base de dados entre os clientes está implementada

como mostra a 3.8. Inicialmente buscamos a base de dados que o Keras disponibiliza e dividimos para cada índice uma porção de dados. Por exemplo: o índice 0 ficaria com as imagens na posição 0 até a posição 5000 dentro do conjunto de treino e de 0 até 1000 no conjunto de validação. Isso é mostrado nas linhas 6 e 7, e 9 e 10 respectivamente em 3.8.

Listing 3.8 – Definição das partições dos clientes

```
1  def load_partition(idx: int):
2      assert idx in range(10)
3      (x_train, y_train), (x_test, y_test) = tf.keras.datasets.
           cifar10.load_data()
4
5      return (
6                  x_train[idx * 5000: (idx + 1) * 5000],
7                  y_train[idx * 5000: (idx + 1) * 5000],
8              ), (
9                  x_test[idx * 1000: (idx + 1) * 1000],
10                 y_test[idx * 1000: (idx + 1) * 1000],
11             )
```

4 Resultados

Este capítulo apresenta os resultados experimentais do cenário montado. Todos os exemplos foram executados com o mesmo ambiente e um teste de cada vez.

4.1 Ambiente

Os experimentos foram feitos em um cluster homogêneo de 7 máquinas da GCP. As máquinas virtuais tinham 4 vCPU, 16gb de RAM e 30gb de SSD no Ubuntu 18.04 como mostrado em 4.1.

Tabela 4.1 – Configuração das máquinas Virtuais

VM	OS	CPU Model	CPU Cores	System RAM	System Storage
1	Ubuntu 18.04	On Demand	4 vCPUs not shared	16GB	30GB
2	Ubuntu 18.04	On Demand	4 vCPUs not shared	16GB	30GB
3	Ubuntu 18.04	On Demand	4 vCPUs not shared	16GB	30GB
4	Ubuntu 18.04	On Demand	4 vCPUs not shared	16GB	30GB
5	Ubuntu 18.04	On Demand	4 vCPUs not shared	16GB	30GB
6	Ubuntu 18.04	On Demand	4 vCPUs not shared	16GB	30GB
7	Ubuntu 18.04	On Demand	4 vCPUs not shared	16GB	30GB

4.2 Experimento

Utilizando o modelo *MobileNetV2* como mencionado em 3.4 e a maneira de agregar os resultados no FL foi a FedAvg descrita em 2.6. O treino realizado nos clientes foi definido em 1 epoca na primeira iteração global do FL e partir da segunda iteração, 2 epocas. Configuramos também 25 iterações globais para o FL. Utilizamos esses parâmetros para limitar o treino em torno de 51 epocas globais. Outro parâmetro variado é a quantidade de passos de avaliação. Definimos 5 epocas até a quarta iteração e 10 para as demais. Estes parâmetros foram adotados, pois são os mesmo utilizados nos exemplos que o *framework Flower* utilizou.

A base de dados treino foi dividida em 10 lotes de 5000 imagens e cada cliente selecionava localmente de maneira aleatória um lote. Durante o treino foi utilizado 20% desse lote para validação e o restante para treino. Os lotes são garantidas de ter pelo menos uma imagem de cada classe, mas não possui o mesmo número de imagens por classe. A base de dados de treinamento foi utilizada em todas as verificações e predições dos testes realizados, com um total de 10.000 imagens na mesma. O tempo utilizado é o tempo fornecido pelo framework *Flower* que é o exato tempo que levou para a conclusão de todas as iterações globais do FL.

Tabela 4.2 – Resultado do caso base

Configuração	Máquinas	Instâncias	Tempo de treinamento	Acurácia	F1
caso base	1	-	9600	0.7415	0.74

Tabela 4.3 – Máquinas x Instâncias por máquinas

Configuração	Máquinas	Instâncias	Tempo de treinamento	Acurácia	F1
1	1	1	1211	0.4691	0.34
2	1	4	2865	0.5583	0.50
3	2	1	1543	0.5248	0.51
4	2	4	3482	0.6168	0.61
5	4	1	1667	0.5406	0.53
6	4	4	5532	0.5940	0.59
7	6	1	1754	0.5889	0.58
8	6	4	6830	0.6430	0.63

Para poder comparar os resultados obtidos nas variações a seguir, executamos um caso base para ter como parâmetro de comparação. O mesmo modelo foi utilizado, com as mesmas 51 epocas. O modelo foi treinado em uma máquina e utilizou todos os 10 lotes para treinar, totalizando 50.000 imagens e as mesmas 10.000 para a base de avaliação e predição. Os resultados obtidos estão na Tabela 4.2. O caso base apresentou o maior tempo de todos os testes, mas em compensação obteve o melhor resultado de acurácia e *F1Score*.

Testamos primeiro variando o número de máquinas e o número de instâncias por máquina. Nestes testes sempre utilizamos 100% das instâncias. Com base na tabela 4.3 percebemos que quando aumentamos o número de instâncias por máquinas o tempo de treinamento aumentava. Isso era esperado, pois aumenta a concorrência por recursos mesmo que em teoria exista um core para cada instância.

A acurácia teve melhores resultados nas configurações com maior número de clientes. Isso pode ser explicado pelo maior número de dados disponível para o treinamento. Entretanto não é suficiente para alcançar o caso base que atingiu 0.7415 e 0.74 de *F1Score* com o mesmo numero de epocas.

Em comparação com o caso base, o aprendizado federado sem nenhum tipo de melhoria não foi suficiente nem mesmo para obter resultados próximos do caso base. Outro ponto explorado foi variar a porcentagem de instâncias selecionadas em cada iteração tanto no passo de treinamento quanto no de avaliação. Para isso fixamos em 4 o número de instâncias por máquinas e variamos a amostra de clientes em 25, 50 e 100 porcento e mantivemos a quantidade de iterações e epochs. A tabela 4.4 mostra os resultados da relação de máquina por amostra de clientes.

O tempo de treinamento demonstrou ser inferior quando se utiliza apenas 25% dos clientes, precisando, em todos os casos, de menos da metade do tempo para concluir as iterações em comparação com o mesmo número de máquinas, mas utilizando 100% dos clientes. Observando todos os resultados, o maior número de máquinas e porcentagem levou mais tempo para concluir

Tabela 4.4 – Amostra x máquinas

Máquinas	Amostra de clientes	Tempo de Treinamento	Acurácia	F1
1	100%	2865	0.5583	0.50
1	50%	1610	0.5191	0.50
1	25%	1398	0.4535	0.42
2	100%	3482	0.6168	0.61
2	50%	1952	0.5281	0.52
2	25%	1623	0.5256	0.52
4	100%	5532	0.5940	0.59
4	50%	6255	0.6284	0.63
4	25%	2741	0.5394	0.53
6	100%	6830	0.6430	0.63
6	50%	3537	0.5912	0.59
6	25%	2063	0.5815	0.57

o que é um comportamento esperado, visto que aumenta a probabilidade de ocorrer duas ou mais instâncias na mesma maquina.

5 Conclusão

Este trabalho compara o tempo de processamento e as métricas de qualidade em várias configurações de aprendizado federado. Apesar de poder ter influência do hardware e da rede, observamos que quando aumentamos a concorrência de clientes em uma mesma máquina o tempo piora em comparação com o mesmo número de clientes em máquinas diferentes. Isso também é observado no caso base, mas não pela quantidade de clientes e sim pela quantidade de dados muito superior aos outros. Isso se mostra coerente com os resultados relatados por (KARIMIREDDY et al., 2019) por conta da deriva de conceito dos modelos. Mas em todas as configurações apresentadas o caso base ainda foi melhor em todas as métricas utilizadas.

Os trabalhos futuros deveriam direcional para combinar técnicas de melhorias tanto no servidor quanto nos clientes visando observar a variação da acurácia em outras configurações. Por outro lado, poderíamos avaliar o impacto do número de clientes no modelo, mas com um número ainda maior de clientes em uma rede menos homogênea.

Referências

SANDLER, Mark and Howard, Andrew and Zhu, Menglong and Zhmoginov, Andrey and Chen, Liang-Chieh. Disponível em: <https://github.com/FedML-AI/FedML/blob/master/research/Awesome-Federated-Learning.md>.

Disponível em: <https://www.ibm.com/br-pt/cloud/learn/neural-networkstoc-tipos-de-r-DbL0dXJo>.

Disponível em: <https://paperswithcode.com>.

Disponível em: <https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec: :text=F1%2Dscore%20when%20Recall%20%3D%201.0%2C%20Precision%20%3D%200.0

ABADI, M.; BARHAM, P.; CHEN, J.; CHEN, Z.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; IRVING, G.; ISARD, M. et al. Tensorflow: A system for large-scale machine learning. In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. [S.l.: s.n.], 2016. p. 265–283.

ANDREUX, M.; TERRAIL, J. O. d.; BEGUIER, C.; TRAMEL, E. W. *Siloed Federated Learning for Multi-Centric Histopathology Datasets*. arXiv, 2020. Disponível em: <https://arxiv.org/abs/2008.07424>.

BEN-NUN, T.; HOEFLER, T. *Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis*. 2018.

BEUTEL, D. J.; TOPAL, T.; MATHUR, A.; QIU, X.; PARCOLLET, T.; LANE, N. D. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

CALDAROLA, D.; CAPUTO, B.; CICCONE, M. *Improving Generalization in Federated Learning by Seeking Flat Minima*. arXiv, 2022. Disponível em: <https://arxiv.org/abs/2203.11834>.

GAVRILOVA, Y. *A Guide to Deep Neural Networks*. Disponível em: <https://serokell.io/blog/deep-learning-and-neural-network-guide#components-of-neural-networks>.

KAIROUZ, P.; MCMAHAN, H. B.; AVENT, B.; BELLET, A.; BENNIS, M.; BHAGOJI, A. N.; BONAWITZ, K.; CHARLES, Z.; CORMODE, G.; CUMMINGS, R.; D'OLIVEIRA, R. G. L.; EICHNER, H.; ROUAYHEB, S. E.; EVANS, D.; GARDNER, J.; GARRETT, Z.; GASCóN, A.; GHAZI, B.; GIBBONS, P. B.; GRUTESER, M.; HARCHAOUI, Z.; HE, C.; HE, L.; HUO, Z.; HUTCHINSON, B.; HSU, J.; JAGGI, M.; JAVIDI, T.; JOSHI, G.; KHODAK, M.; KONEčNý, J.; KOROLOVA, A.; KOUSHANFAR, F.; KOYEJO, S.; LEPOINT, T.; LIU, Y.; MITTAL, P.; MOHRI, M.; NOCK, R.; ÖZGüR, A.; PAGH, R.; RAYKOVA, M.; QI, H.; RAMAGE, D.; RASKAR, R.; SONG, D.; SONG, W.; STICH, S. U.; SUN, Z.; SURESH, A. T.; TRAMèR, F.; VEPAKOMMA, P.; WANG, J.; XIONG, L.; XU, Z.; YANG, Q.; YU, F. X.; YU, H.; ZHAO, S. *Advances and Open Problems in Federated Learning*. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1912.04977>.

KARIMIREDDY, S. P.; KALE, S.; MOHRI, M.; REDDI, S. J.; STICH, S. U.; SURESH, A. T. *SCAFFOLD: Stochastic Controlled Averaging for Federated Learning*. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1910.06378>.

KERAS. *Keras*. 2021. Available at: <https://en.wikipedia.org/wiki/Keras>.

KESKAR, N. S.; MUDIGERE, D.; NOCEDAL, J.; SMELYANSKIY, M.; TANG, P. T. P. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. arXiv, 2016. Disponível em: <https://arxiv.org/abs/1609.04836>.

KRIZHEVSKY, A. *Learning multiple layers of features from tiny images*. [S.l.], 2009.

LIU, Q.; CHEN, C.; QIN, J.; DOU, Q.; HENG, P.-A. *FedDG: Federated Domain Generalization on Medical Image Segmentation via Episodic Learning in Continuous Frequency Space*. arXiv, 2021. Disponível em: <https://arxiv.org/abs/2103.06030>.

MCMAHAN, H. B.; MOORE, E.; RAMAGE, D.; HAMPSON, S.; ARCAS, B. A. y. Communication-efficient learning of deep networks from decentralized data. arXiv, 2016. Disponível em: <https://arxiv.org/abs/1602.05629>.

NGUYEN, D. C.; PHAM, Q.-V.; PATHIRANA, P. N.; DING, M.; SENEVIRATNE, A.; LIN, Z.; DOBRE, O. A.; HWANG, W.-J. *Federated Learning for Smart Healthcare: A Survey*. arXiv, 2021. Disponível em: <https://arxiv.org/abs/2111.08834>.

NIELSEN, M. A. *Neural Networks and Deep Learning*. 2016. Disponível em: <http://neuralnetworksanddeeplearning.com>.

ROJAS, R. *Neural networks: a systematic introduction*. [S.l.]: Springer Science & Business Media, 2013.

RUDER, S. *An overview of gradient descent optimization algorithms*. arXiv, 2016. Disponível em: <https://arxiv.org/abs/1609.04747>.

SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A.; CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1801.04381>.

SMITH, V.; CHIANG, C.-K.; SANJABI, M.; TALWALKAR, A. *Federated Multi-Task Learning*. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1705.10467>.