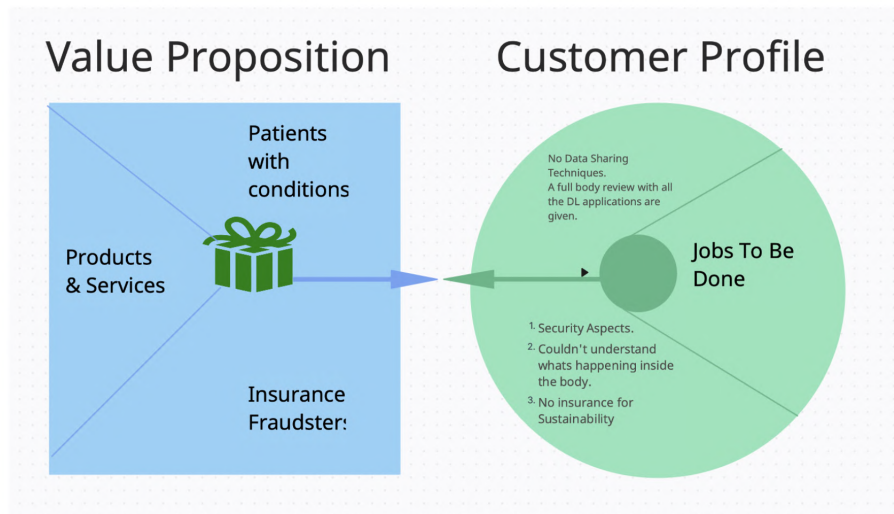


Chapter 2

Minimum Viable Product

2.1 Ideation Principles

Developing a Minimum Viable Product (MVP) involves addressing the root cause of privacy concerns in IoT devices. The focus is on countering the alarming statistic from market research, where 90% of Deep Learning (DL) models fail to reach production. To tackle issues like data leaks to pharmaceutical companies for advertisements, the MVP aims to optimize Edge devices. The strategy involves making communication lightweight, ensuring it occurs when the device is at rest, facilitating the deployment of new weights. This streamlined approach addresses core problems while laying the foundation for a robust and privacy-preserving IoT solution.



2.2 Minimum Viable Features

Our primary goal is to deliver value quickly, test the product in the market, and learn from user interactions. An MVP helps in minimizing development time and resources while

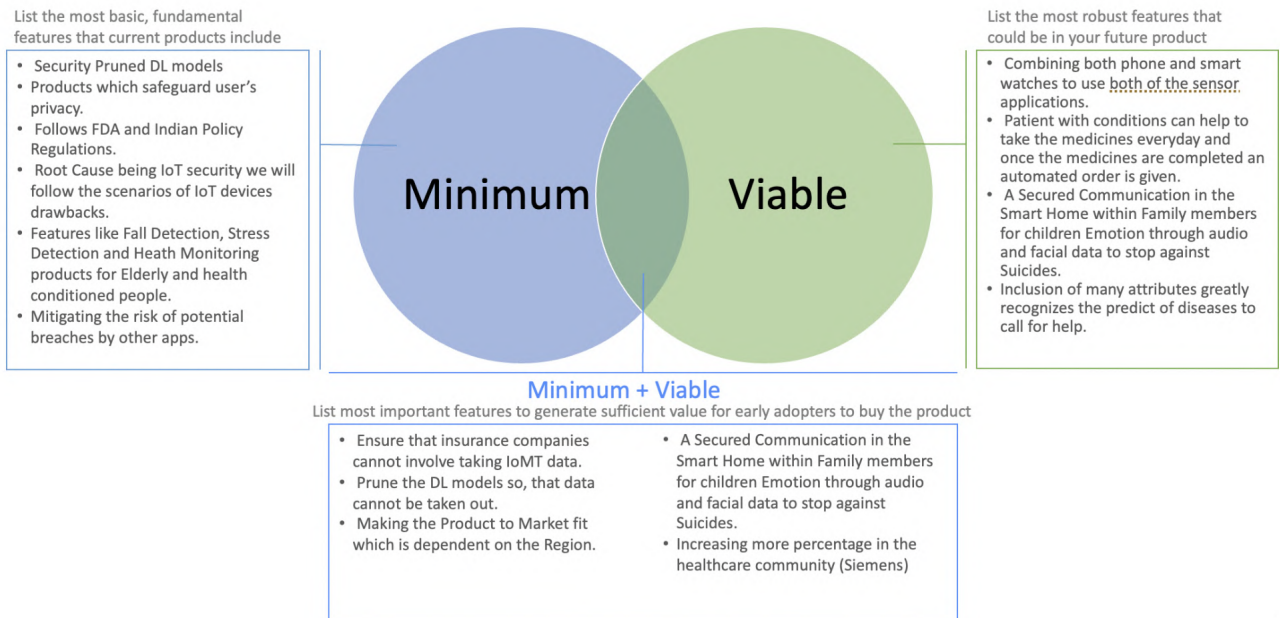
allowing for rapid iteration based on user responses.

2.2.1 Robust with Fundamental Features

The Security Pruned DL models focus on safeguarding user privacy, adhering to FDA and Indian Policy Regulations. Addressing IoT security concerns, the product targets scenarios of IoT device drawbacks. Key features include Fall Detection, Stress Detection, and Health Monitoring for elderly and health-conditioned individuals, mitigating the risk of breaches by other apps.

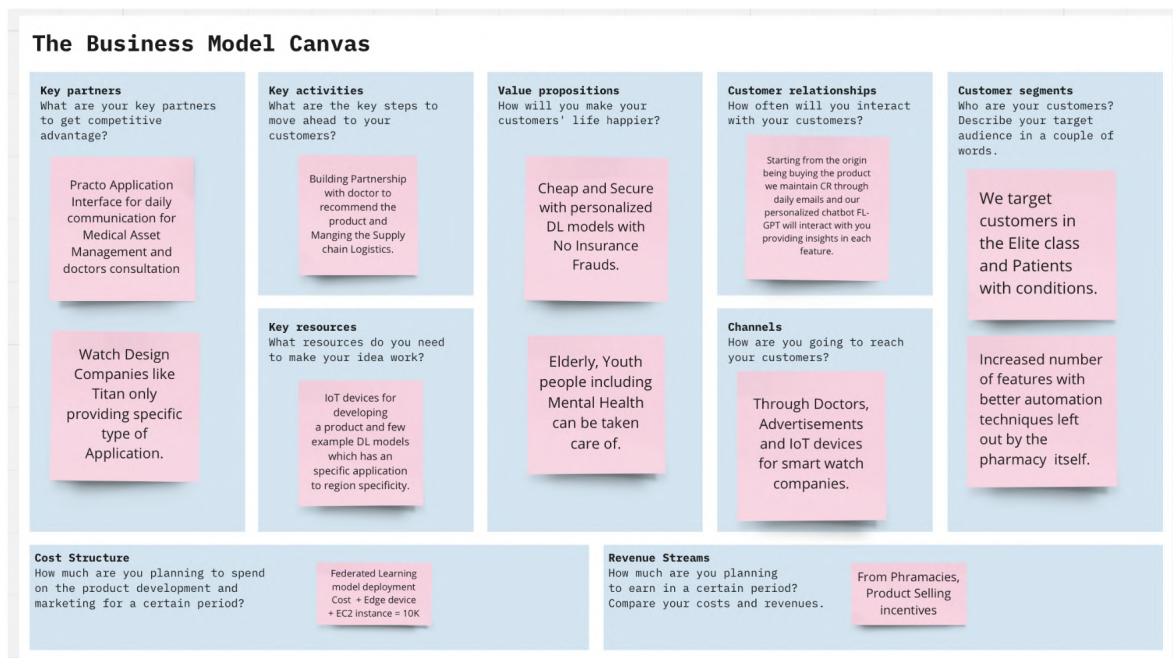
2.2.2 Minimum + Viable

For early adopters, the MVP emphasizes features crucial for generating sufficient value. Ensuring insurance companies cannot access IoT data, pruning DL models to prevent data extraction, and tailoring the product for regional market fit are paramount. The integration of both phone and smartwatch sensors for comprehensive applications, automated medicine orders for patients with conditions, and a secure communication platform within smart homes for emotion detection contribute to a robust and market-worthy product. Increasing engagement in the healthcare community, exemplified by partnerships with companies like Siemens, further strengthens the product's viability.



Chapter 3

Business Model Development



Key Partners

To gain a competitive advantage, our key partners include the Practo Application Interface for daily communication, facilitating Medical Asset Management, and enabling doctor consultations.

Key Activities

The key steps to move ahead to our customers involve building partnerships with doctors to recommend our product and efficiently managing the supply chain logistics. Additionally,

designing the watch interface with companies like Titan ensures a tailored application.

Key Resources

To make our idea work, we need IoT devices for product development and a collection of example DL models with region-specific applications.

Value Propositions

We aim to make our customers' lives happier by providing a cheap and secure solution with personalized DL models, eliminating insurance fraud risks.

Customer Relationships

We maintain continuous interaction with customers, starting from product purchase. Daily emails and our personalized chatbot FL-GPT offer insights into each feature.

Customer Segments

Our target customers include the elite class, patients with conditions, and elderly individuals. We extend our services to youth, focusing on mental health care.

Channels

We reach our customers through doctors, advertisements, and IoT devices for smartwatch companies.

Revenue Streams

We plan to earn revenue through increased features and automation techniques, left out by pharmacies. Product selling incentives from pharmacies also contribute to our revenue streams.

3.1 Cost Structure

Configure Amazon EC2

Info

×

Estimated commitment price based on the following selections:
Instance type: **t4g.nano** Operating system: **Linux**

Select the container and options to find your best price

☒ **Compute Savings Plans**

One plan that automatically applies to all usage on EC2, Fargate, and Lambda. Up to 66% discount. [Learn more](#)

Reservation term

☐ 1 year
☒ 3 year

Payment Options

☒ No upfront
☐ Partial upfront
☐ All upfront

Upfront: 0.00
Monthly: 1.02/Month

☐ **EC2 Instance Savings Plans**

Get deeper discount when you only need one instance family and region. Up to 72% discount. [Learn more](#)

Reservation term

☐ 1 year
☒ 3 year

Payment Options

☒ No upfront
☐ Partial upfront
☐ All upfront

Upfront: 0.00
Monthly: 0.88/Month

☐ **On-Demand**

Maximize flexibility. [Learn more](#)

Expected utilization

Enter the expected usage of Amazon EC2 instances

Usage

100

Usage type

Utilization percent per month

Instance: 0.0028/Hour
Monthly: 2.04/Month

☐ **Spot Instances**

Minimize cost by leveraging EC2's spare capacity. Recommended for fault tolerant and interruption tolerant applications. [Learn more](#)

The historical average discount for t4g.nano is 64%

Assume percentage discount for my estimate

64

Actual spot instance pricing varies

With spot instances, you pay the spot price that's in effect for the time period your instance is running

Instance: 0.0028/Hour
Monthly: 0.74/Month

► Other purchasing options

Total Upfront cost: 0.00 USD
Total Monthly cost: 1.02 USD

Show Details

Save and view summary

Save and add service

8

Chapter 4

Business Plan and Access to Funding

4.1 Executive Summary

Expanding our focus on the Internet of Medical Things, our primary target is individuals dealing with health issues and patients. However, we aim to go beyond by introducing innovative applications:

1. **Smart Doctor Integration:** Our system will not only monitor health but also act as a Smart Doctor. It will provide personalized health recommendations and facilitate seamless connections with healthcare platforms like Practo, ensuring direct delivery of prescribed medicines.
2. **Emotional Friendliness:** Recognizing the prevalence of mental health issues, especially in America, our model will incorporate emotional friendliness. By understanding and addressing negative consequences, we aspire to offer support and comfort, contributing to a holistic approach to well-being.

4.2 Company Description

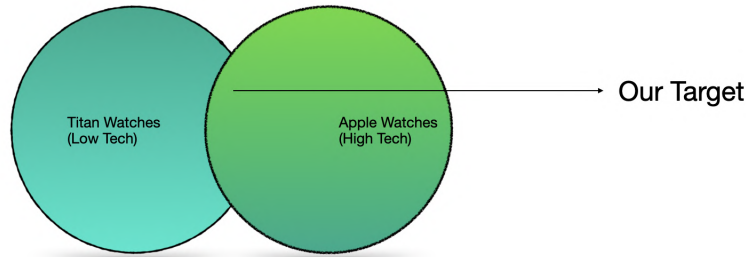
An IoMT startup working towards the goal for...

1. **Privacy Preserving DL models:** Investigating the level of privacy required for unrestricted use of an individual's health data across various applications is a critical research question. If privacy is guaranteed at 100%, what is the potential number of applications that could harness a person's health data?
2. **Continuous Learning:** Our vision involves perpetual learning models, eliminating the need for recurrent training on new data. This approach ensures that our systems stay updated and relevant without the necessity for constant retraining.
3. **Rich Personalization with Affordable Cost:** We aim to achieve extensive personalization in our IoMT applications while maintaining an affordable cost structure. This includes providing advanced features similar to premium products like the Apple Watch but at a more accessible and budget-friendly price point.

4.3 Market Research

BUSINESS PLAN

Market Research



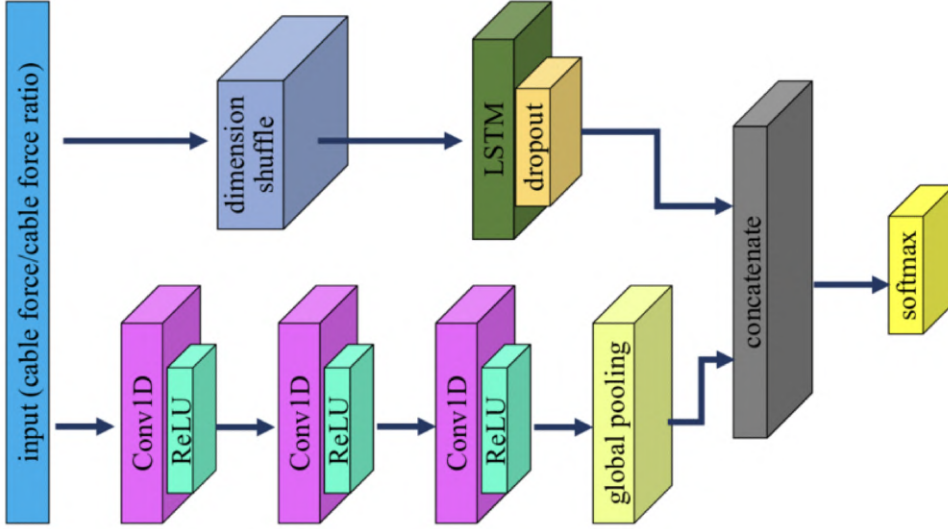
4.4 Competitive Analysis

In order for our product to survive in the market, we adopt a strategic approach:

1. **Focus on Healthcare Professionals:** Our initial emphasis is on doctors, and we invest in compensating them to endorse and recommend our devices. This strategic partnership ensures that our main goal of providing essential healthcare solutions, especially for the elderly, is effectively promoted.
2. **Diverse Applications with Performance and Battery Life Optimization:** We strive to include a diverse set of applications in our devices. By enhancing performance and optimizing battery life, we ensure that our devices cater to a broad range of user needs. This approach allows us to address various health concerns and scenarios, making our product more versatile in the market.

4.5 Architecture of Our Model

Although we have selected the minimal applications to show that when connected these devices to the internet or through a phone to decrease the vulnerability we use Federated Learning to improve the Personalization, Recommendations and User Experience.



4.6 Execution Plan (Case Study : HAR) Output

We implemented Human Activity Recognition (HAR) using the LSTM-FCN model, combining Long Short-Term Memory (LSTM) and Fully Convolutional Networks (FCN). It classifies activities like walking, sitting, etc., based on inertial sensor data. The model is trained using federated learning across multiple clients and achieves a commendable 80% accuracy on the test data, showcasing its effectiveness in privacy-preserving machine learning for activity recognition.


```
...thon3.10 ~/miniconda3/envs/tensorflow/bin/jupyter-notebook • python ... ..esktop/Programs/Federated Learning/IoMT/stress_detection -- -zsh ... ../Federated Learning/IoMT/activity_recognition -- python3 main.py +
Epoch 2/10
39/39 [=====] - 3s 78ms/step - loss: 0.3428 - accuracy: 0.8874
Epoch 3/10
39/39 [=====] - 3s 78ms/step - loss: 0.2020 - accuracy: 0.9315
Epoch 4/10
39/39 [=====] - 3s 79ms/step - loss: 0.1484 - accuracy: 0.9486
Epoch 5/10
39/39 [=====] - 3s 78ms/step - loss: 0.1548 - accuracy: 0.9421
Epoch 6/10
39/39 [=====] - 3s 78ms/step - loss: 0.1135 - accuracy: 0.9506
Epoch 7/10
39/39 [=====] - 3s 79ms/step - loss: 0.1056 - accuracy: 0.9576
Epoch 8/10
39/39 [=====] - 3s 79ms/step - loss: 0.1120 - accuracy: 0.9527
Epoch 9/10
39/39 [=====] - 3s 78ms/step - loss: 0.1064 - accuracy: 0.9539
Epoch 10/10
39/39 [=====] - 3s 78ms/step - loss: 0.1062 - accuracy: 0.9514
Client 0 - Test Accuracy: 0.74, Test Loss: 0.8616
Epoch 1/10
39/39 [=====] - 22s 188ms/step - loss: 1.0857 - accuracy: 0.6548
Epoch 2/10
39/39 [=====] - 3s 81ms/step - loss: 0.4704 - accuracy: 0.8413
Epoch 3/10
39/39 [=====] - 3s 82ms/step - loss: 0.2831 - accuracy: 0.9066
Epoch 4/10
39/39 [=====] - 3s 82ms/step - loss: 0.1729 - accuracy: 0.9449
Epoch 5/10
39/39 [=====] - 3s 82ms/step - loss: 0.1606 - accuracy: 0.9429
Epoch 6/10
39/39 [=====] - 3s 82ms/step - loss: 0.1391 - accuracy: 0.9506
Epoch 7/10
39/39 [=====] - 3s 83ms/step - loss: 0.1276 - accuracy: 0.9523
Epoch 8/10
39/39 [=====] - 3s 81ms/step - loss: 0.1044 - accuracy: 0.9559
Epoch 9/10
39/39 [=====] - 3s 82ms/step - loss: 0.1211 - accuracy: 0.9494
Epoch 10/10
39/39 [=====] - 3s 81ms/step - loss: 0.1180 - accuracy: 0.9547
Client 1 - Test Accuracy: 0.71, Test Loss: 0.9912
Epoch 1/10
2023-11-02 04:03:46.951696: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:961] model_pruner failed: INVALID_ARGUMENT: Graph does not contain terminal node Adam/Assign
AddVariableOp.
39/39 [=====] - 24s 203ms/step - loss: 0.7220 - accuracy: 0.7449
Epoch 2/10
39/39 [=====] - 3s 83ms/step - loss: 0.2412 - accuracy: 0.9273
Epoch 3/10
39/39 [=====] - 3s 82ms/step - loss: 0.1685 - accuracy: 0.9449
Epoch 4/10
39/39 [=====] - 3s 82ms/step - loss: 0.1419 - accuracy: 0.9510
```

4.7 Execution Plan (Case Study : HR Stress) Output

We implemented personal heart rate datasets for federated learning, employing a global BiLSTM (Bidirectional Long Short-Term Memory) and LSTM- FCN neural network model. The global model is trained on 26 individual client datasets, demonstrating robustness and privacy preservation. After training, the model achieves an impressive 94% accuracy on the test data, showcasing the efficacy of the BiLSTM architecture in capturing temporal dependencies for heart rate classification.

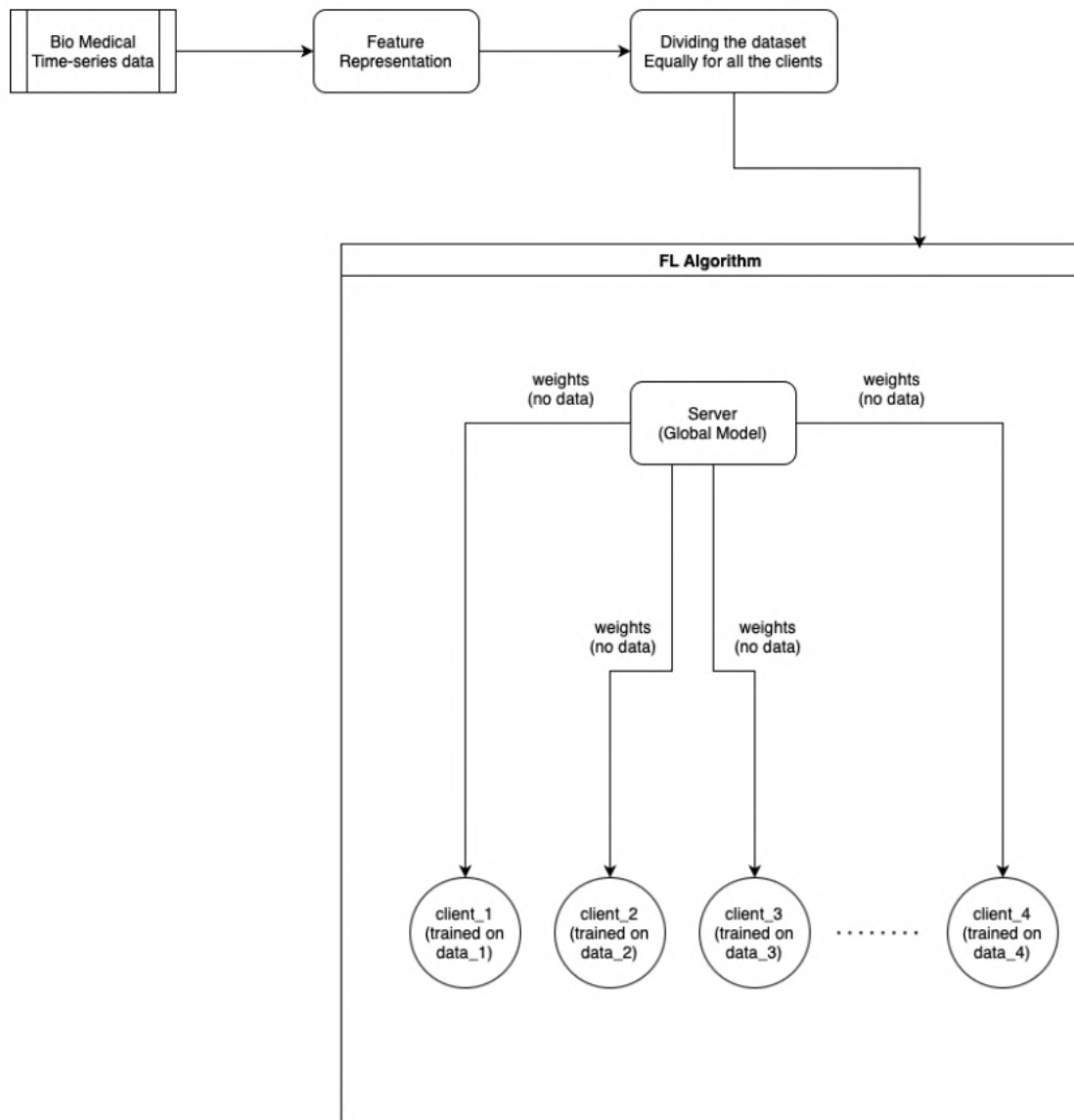
```

...thon3.10 ~/miniconda3/envs/tensorflow/bin/jupyter-notebook • python ... ..esktop/Programs/Federated Learning/IoMT/stress_detection -- -zsh .../Federated Learning/IoMT/activity_recognition -- python3 main.py +
Epoch 10/10
2/2 [=====] - 0s 10ms/step - loss: -13.3378 - accuracy: 0.8836
1/1 [=====] - 1s 996ms/step - loss: -14.1987 - accuracy: 0.8000
Dataset 24 - Test Loss: -14.198735237121582, Test Accuracy: 0.800000011920929
Training on dataset 25...
Epoch 1/10
2/2 [=====] - 4s 1s/step - loss: 19.9840 - accuracy: 0.0000e+00
Epoch 2/10
2/2 [=====] - 0s 30ms/step - loss: 8.2454 - accuracy: 0.8588
Epoch 3/10
2/2 [=====] - 0s 18ms/step - loss: 0.7304 - accuracy: 0.6078
Epoch 4/10
2/2 [=====] - 0s 17ms/step - loss: -2.9063 - accuracy: 0.8039
Epoch 5/10
2/2 [=====] - 0s 10ms/step - loss: -5.1672 - accuracy: 0.8039
Epoch 6/10
2/2 [=====] - 0s 16ms/step - loss: -7.1355 - accuracy: 0.8039
Epoch 7/10
2/2 [=====] - 0s 10ms/step - loss: -8.9749 - accuracy: 0.8235
Epoch 8/10
2/2 [=====] - 0s 8ms/step - loss: -11.0057 - accuracy: 0.8235
Epoch 9/10
2/2 [=====] - 0s 17ms/step - loss: -12.4965 - accuracy: 0.8235
Epoch 10/10
2/2 [=====] - 0s 10ms/step - loss: -14.1272 - accuracy: 0.8235
1/1 [=====] - 1s 1s/step - loss: -7.3749 - accuracy: 0.9231
Dataset 25 - Test Loss: -7.374940872192383, Test Accuracy: 0.9230769276618958
Training on dataset 26...
Epoch 1/10
2/2 [=====] - 4s 1s/step - loss: 0.5529 - accuracy: 0.6333
Epoch 2/10
2/2 [=====] - 0s 17ms/step - loss: -2.2633 - accuracy: 0.8000
Epoch 3/10
2/2 [=====] - 0s 16ms/step - loss: -4.3514 - accuracy: 0.8000
Epoch 4/10
2/2 [=====] - 0s 9ms/step - loss: -6.1854 - accuracy: 0.8000
Epoch 5/10
2/2 [=====] - 0s 18ms/step - loss: -7.8718 - accuracy: 0.8000
Epoch 6/10
2/2 [=====] - 0s 18ms/step - loss: -9.9419 - accuracy: 0.8000
Epoch 7/10
2/2 [=====] - 0s 10ms/step - loss: -11.9485 - accuracy: 0.8000
Epoch 8/10
2/2 [=====] - 0s 15ms/step - loss: -13.6870 - accuracy: 0.8000
Epoch 9/10
2/2 [=====] - 0s 8ms/step - loss: -15.4758 - accuracy: 0.8000
Epoch 10/10
2/2 [=====] - 0s 16ms/step - loss: -17.1462 - accuracy: 0.8000
1/1 [=====] - 1s 1s/step - loss: -18.9692 - accuracy: 0.8000
Dataset 26 - Test Loss: -18.96924591064453, Test Accuracy: 0.800000011920929

```

Chapter 5

Proof of Concept



Chapter 6

Python Implementation

6.1 Libraries

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import os
4 from patchify import patchify
5 import cv2
6 from keras.models import load_model
7 import random
8 from PIL import Image
9
10
11
12 #Imports
13 import os
14 from google.colab import drive
15 from PIL import Image
16 import os
17 from pathlib import Path
18 from google.auth.transport.requests import AuthorizedSession
19 from google.oauth2 import service_account
20 from pprint import pprint
21 import json
22 import ee
23 from IPython.display import Image
24
25
26 import matplotlib.pyplot as plt
27 import numpy as np
28 import os
29 from keras.models import load_model
30 from keras.preprocessing.image import ImageDataGenerator
31 import cv2
32 from sklearn.model_selection import train_test_split
33 from sklearn.preprocessing import MinMaxScaler
34 from keras.callbacks import ModelCheckpoint, LearningRateScheduler,
    EarlyStopping
```

```

35 import random
36
37
38
39 import cv2
40 import numpy as np
41 import os
42 import matplotlib.pyplot as plt
43 from PIL import Image
44 from patchify import patchify
45 import splitfolders
46 import random
47 from keras.utils import to_categorical

```

6.2 Code

6.2.1 Human Stress Recognition

```

1
2 # Load and preprocess your datasets
3 datasets = [
4     pd.read_csv('./personal_HR/1.csv'),
5     pd.read_csv('./personal_HR/2.csv'),
6     pd.read_csv('./personal_HR/3.csv'),
7     pd.read_csv('./personal_HR/4.csv'),
8     pd.read_csv('./personal_HR/5.csv'),
9     pd.read_csv('./personal_HR/6.csv'),
10    pd.read_csv('./personal_HR/7.csv'),
11    pd.read_csv('./personal_HR/8.csv'),
12    pd.read_csv('./personal_HR/9.csv'),
13    pd.read_csv('./personal_HR/10.csv'),
14    pd.read_csv('./personal_HR/11.csv'),
15    pd.read_csv('./personal_HR/12.csv'),
16    pd.read_csv('./personal_HR/13.csv'),
17    pd.read_csv('./personal_HR/14.csv'),
18    pd.read_csv('./personal_HR/15.csv'),
19    pd.read_csv('./personal_HR/16.csv'),
20    pd.read_csv('./personal_HR/17.csv'),
21    pd.read_csv('./personal_HR/18.csv'),
22    pd.read_csv('./personal_HR/19.csv'),
23    pd.read_csv('./personal_HR/20.csv'),
24    pd.read_csv('./personal_HR/21.csv'),
25    pd.read_csv('./personal_HR/22.csv'),
26    pd.read_csv('./personal_HR/23.csv'),
27    pd.read_csv('./personal_HR/24.csv'),
28    pd.read_csv('./personal_HR/25.csv'),
29    pd.read_csv('./personal_HR/26.csv')
30 ]
31
32 # Create and compile your Keras model
33 global_model = Sequential()
34 global_model.add(Dense(64, input_dim=16, activation='relu'))

```

```

35 global_model.add(Dense(32, activation='relu'))
36 global_model.add(Dense(1, activation='sigmoid'))
37
38
39
40
41
42
43 num_clients = 26
44
45
46 test_loss_clients = []
47 test_accuracy_clients = []
48 clients_weights = []
49 client_models = []
50
51 # Training and evaluation loop
52 for i, dataset in enumerate(datasets):
53     print(f"Training on dataset {i + 1}...")
54
55     # Preprocess the dataset
56     X = dataset.drop(columns=["class"]).values
57     y = dataset["class"].values
58
59     model = Sequential()
60     model.add(Dense(64, input_dim=16, activation='relu'))
61     model.add(Dense(32, activation='relu'))
62     model.add(Dense(1, activation='sigmoid'))
63     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
accuracy'])
64
65
66     # Split the data into training and testing sets
67     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.2, random_state=42)
68
69     # Train the model on the dataset
70     model.fit(X_train, y_train, epochs=10, batch_size=32)
71
72     # Evaluate the model on the test data
73     loss, accuracy = model.evaluate(X_test, y_test)
74     test_loss_clients.append(loss)
75     test_accuracy_clients.append(loss)
76     client_models.append(model)
77     print(f"Dataset {i + 1} - Test Loss: {loss}, Test Accuracy: {accuracy}
")
78
79
80
81
82 global_weights = global_model.get_weights()
83 for i in range(len(clients_weights)):
84     for j in range(num_clients):
85         client_weights = client_models[j].get_weights()

```

```

86         global_weights[i] += client_weights[i]
87
88
89
90
91
92 global_model.set_weights(global_weights)
93
94
95 global_model.compile(loss='binary_crossentropy', optimizer='adam', metrics
    =['accuracy'])
96
97
98
99 for i, dataset in enumerate(datasets):
100     print(f"Global Model testing on dataset {i + 1}...")
101
102     # Preprocess the dataset
103     X = dataset.drop(columns=["class"]).values
104     y = dataset["class"].values
105     # Evaluate the global model
106     test_loss, test_accuracy = global_model.evaluate(X, y, verbose=0)
107     print(f'Global Test Accuracy: {test_accuracy:.2f}, Global Test Loss: {
        test_loss:.4f}')

```

6.2.2 Human Activity Recognition

```

1
2 import numpy as np
3 import matplotlib
4 import matplotlib.pyplot as plt
5 import tensorflow as tf # Version 1.0.0 (some previous versions are used
    in past commits)
6 from sklearn import metrics
7 import os
8 import tensorflow as tf
9 from tensorflow import keras
10 from tensorflow.keras.layers import LSTM, Dense, Dropout
11 from tensorflow.keras.layers import LSTM, Dense, BatchNormalization
12 from tensorflow.keras.optimizers import RMSprop
13 from tensorflow.keras.layers import LSTM, Dense, BatchNormalization
14
15 # Useful Constants
16 DATA_PATH = "data/"
17 DATASET_PATH = DATA_PATH + "UCI HAR Dataset/"
18 print("\n" + "Dataset is now located at: " + DATASET_PATH)
19
20 # Those are separate normalised input features for the neural network
21 INPUT_SIGNAL_TYPES = [
22     "body_acc_x_",
23     "body_acc_y_",
24     "body_acc_z_",

```

```

25     "body_gyro_x_",
26     "body_gyro_y_",
27     "body_gyro_z_",
28     "total_acc_x_",
29     "total_acc_y_",
30     "total_acc_z_"
31 ]
32
33 # Output classes to learn how to classify
34 LABELS = [
35     "WALKING",
36     "WALKING_UPSTAIRS",
37     "WALKING_DOWNSTAIRS",
38     "SITTING",
39     "STANDING",
40     "LAYING"
41 ]
42
43
44
45 TRAIN = "train/"
46 TEST = "test/"
47
48
49 # Load "X" (the neural network's training and testing inputs)
50
51 def load_X(X_signals_paths):
52     X_signals = []
53
54     for signal_type_path in X_signals_paths:
55         file = open(signal_type_path, 'r')
56         # Read dataset from disk, dealing with text files' syntax
57         X_signals.append(
58             [np.array(serie, dtype=np.float32) for serie in [
59                 row.replace(' ', ' ').strip().split(' ') for row in file
60             ]]
61         )
62         file.close()
63
64     return np.transpose(np.array(X_signals), (1, 2, 0))
65
66
67
68
69
70 X_train_signals_paths = [
71     DATASET_PATH + TRAIN + "Inertial Signals/" + signal + "train.txt" for
72     signal in INPUT_SIGNAL_TYPES
73 ]
74 X_test_signals_paths = [
75     DATASET_PATH + TEST + "Inertial Signals/" + signal + "test.txt" for
76     signal in INPUT_SIGNAL_TYPES
77 ]

```



```

77 X_train = load_X(X_train_signals_paths)
78 X_test = load_X(X_test_signals_paths)
79
80
81
82
83 # Load "y" (the neural network's training and testing outputs)
84
85 def load_y(y_path):
86     file = open(y_path, 'r')
87     # Read dataset from disk, dealing with text file's syntax
88     y_ = np.array(
89         [elem for elem in [
90             row.replace(' ', ' ').strip().split(' ') for row in file
91         ]],
92         dtype=np.int32
93     )
94     file.close()
95
96     # Subtract 1 to each output class for friendly 0-based indexing
97     return y_ - 1
98
99 y_train_path = DATASET_PATH + TRAIN + "y_train.txt"
100 y_test_path = DATASET_PATH + TEST + "y_test.txt"
101
102 y_train = load_y(y_train_path)
103 y_test = load_y(y_test_path)
104
105
106
107 # Input Data
108
109 training_data_count = len(X_train) # 7352 training series (with 50%
    overlap between each serie)
110 test_data_count = len(X_test) # 2947 testing series
111 n_steps = len(X_train[0]) # 128 timesteps per series
112 n_input = len(X_train[0][0]) # 9 input parameters per timestep
113
114
115 # LSTM Neural Network's internal structure
116 n_hidden = 32 # Hidden layer num of features
117 n_classes = 6 # Total classes (should go up, or should go down)
118
119
120 # Training
121 learning_rate = 0.01
122 lambda_loss_amount = 0.0015
123 training_iters = training_data_count * 300 # Loop 300 times on the
    dataset
124 batch_size = 1500
125 display_iter = 30000 # To show test set accuracy during training
126
127
128 # Some debugging info

```

```

129
130 print("Some useful info to get an insight on dataset's shape and
    normalisation:")
131 print("(X shape, y shape, every X's mean, every X's standard deviation)")
132 print(X_test.shape, y_test.shape, np.mean(X_test), np.std(X_test))
133 print("The dataset is therefore properly normalised, as expected, but not
    yet one-hot encoded.")
134
135
136
137
138 def LSTM_RNN(input_shape, n_hidden, n_classes):
139     model = keras.Sequential()
140     model.add(LSTM(n_hidden, return_sequences=True, input_shape=
input_shape))
141     model.add(BatchNormalization())
142     model.add(LSTM(n_hidden, return_sequences=True))
143     model.add(BatchNormalization())
144     model.add(LSTM(n_hidden, return_sequences=True))
145     model.add(BatchNormalization())
146     model.add(LSTM(n_hidden))
147     model.add(BatchNormalization())
148     model.add(Dense(n_classes, activation='softmax'))
149     return model
150
151
152
153 # Training parameters
154 batch_size = 64
155 num_epochs = 10
156 num_clients = 3
157
158
159
160
161
162 # Define the model and placeholders
163 n_input = 9
164 n_steps = 128 # The shape of your input data
165
166 n_hidden = 32
167 n_classes = 6
168 input_shape = (n_steps, n_input) # Define the global input shape
169
170 X_train_clients = np.array_split(X_train, num_clients)
171 y_train_clients = np.array_split(y_train, num_clients)
172 X_test_clients = np.array_split(X_test, num_clients)
173 y_test_clients = np.array_split(y_test, num_clients)
174
175 test_loss_clients = []
176 test_accuracy_clients = []
177 clients_weights = []
178 client_models = []
179 for i in range(num_clients):

```

```

180     model = LSTM_RNN(input_shape, n_hidden, n_classes)
181     model.compile(optimizer='adam', loss='categorical_crossentropy',
182 metrics=['accuracy'])
182     y_train_one_hot = keras.utils.to_categorical(y_train_clients[i],
183 n_classes)
183     model.fit(X_train_clients[i], y_train_one_hot, batch_size=batch_size,
184 epochs=num_epochs, verbose=1)
184     test_loss, test_accuracy = model.evaluate(X_test_clients[i], keras.
185 utils.to_categorical(y_test_clients[i], n_classes), verbose=0)
185     test_loss_clients.append(test_loss)
186     test_accuracy_clients.append(test_accuracy)
187     clients_weights.append(model.get_weights())
188     client_models.append(model)
189     print(f'Client {i} - Test Accuracy: {test_accuracy:.2f}, Test Loss: {
190 test_loss:.4f}')
190
191 global_model = LSTM_RNN(input_shape, n_hidden, n_classes)
192
193
194
195
196
197 global_weights = global_model.get_weights()
198 for i in range(len(clients_weights)):
199     for j in range(num_clients):
200         client_weights = client_models[j].get_weights()
201         global_weights[i] += client_weights[i]
202
203 global_model.set_weights(global_weights)
204 y_train_one_hot = keras.utils.to_categorical(y_train, n_classes)
205 # Compile the global model
206 global_model.compile(optimizer='adam', loss='categorical_crossentropy',
207 metrics=['accuracy'])
207 global_model.fit(X_train, y_train_one_hot, batch_size=batch_size, epochs=
208 num_epochs, verbose=1)
208
209 # Evaluate the global model
210 test_loss, test_accuracy = global_model.evaluate(X_test_clients[0], keras.
211 utils.to_categorical(y_test_clients[0], n_classes), verbose=0)
211 print(f'Global Test Accuracy: {test_accuracy:.2f}, Global Test Loss: {
212 test_loss:.4f}')

```