ENHANCING ENERGY EFFICIENCY WITH MACHINE LEARNING

Master of Engineering in Information Technology

Individual Project

Kirankumar Athirala
Mat No: 1384194
kirankumar.athirala@stud.fra-uas.de

Abstract*-- The project is basically the implementation of a Long Short-Term Memory model capable of predicting the presence of a human in a fixed environment using ultrasonic sensor data. The model will determine patterns presented by input data which are in line with the possibility of human activity using advanced sequence learning capabilities of LSTM. In this work, we show the potential of the model in making accurate predictions of instances that reflect human presence or absence through a holistic pipeline of data preprocessing, model training, and careful evaluation. The outcomes from the project point out that the LSTM model is able to capture time relationships and therefore has great performance to apply in systems such as surveillance or smart home.

Keywords: Long Short-Term Memory (LSTM), Human Presence Detection, Time-Series Analysis, Sequence Learning, Model Evaluation, Predictive Analytics, Smart Home Systems, Surveillance, Occupancy Detection, Human-Activity Recognition, Energy Efficiency.

I. INTRODUCTION

The need for precision detection of human presence is therefore felt to be growingly necessary in places as different as from domestic automation to security infrastructures[1], in view of the sensing ability of sensor technology advances and data analyses capabilities on the horizon. Recently, the Long Short-Term Memory (LSTM) neural networks have come into consideration due to their capability in time series data analysis for capturing long-term dependencies of data. These networks portrayed a strong capability of understanding complex patterns emanating from sensors and cameras. They are clearly better in precision at moving object detection than any traditional methodologies of detection.

LSTM models naturally deal with many challenges occurring in human detection, like differences in activity level, environmental conditions, and sensor noise. Their great adaptability to variations in the sequences and strong ability to filter out irrelevant data make them highly relevant for this domain. The result, therefore, is that LSTM models plays a essential role in the continuing improvement of the human detection technology, paving the way for advanced and intuitive solutions in the field of monitoring.

II. METHODOLOGY

1. Data Collection:

A. Equipment Setup

An ultrasonic sensor was strategically mounted on top of a stand. The sensor's orientation was pointing downwards to accurately capture data based on the proximity of objects below it.

Figure 1 Experiment Setup

B. Data Collection Environment

The environment was controlled to minimize external interference. The primary subject of data collection was a person positioned in various stances to simulate different real-world scenarios. These stances included sitting on a chair directly beneath the sensor and standing in predetermined positions relative to the sensor: left, right, back, and front of the sensor.

Figure 2 Data Collection with different positions

C. Procedure

The ultrasonic sensor was used to emit sound waves that, upon hitting the subject, would reflect to the sensor. The time taken for the echoes to return was recorded for each stance. This process enabled the collection of data representing the distance between the sensor and the subject in different positions. The data was captured multiple times for each position to ensure consistency and accuracy.

2. Model Architecture

The LSTM extends the memory of the Recurrent Neural Network (RNN) to learn patterns from temporal sequential input[2]. It has been used to interpret sensor data from a smart home for human activity recognition[3], [4]. Unlike RNN, LSTM addresses the vanishing gradient problem, allowing it to learn long-term sequences as well as the influence of initial dependencies. LSTM has been widely employed for applications that need temporal dependence between data [5], such as natural language processing [6], stock market prediction [7], and speech recognition [8].

Each LSTM contains three gates forget, input, and output, which delete or add information to the cell state. The cell state is the most important component of LSTMs since it stores and transmits information between them. Figure 3 depicts how the three gates are connected to the cell state and to one another. Each LSTM cell functions as a memory, erasing, reading, and writing information dependent on the outcomes of the forget, output, and input gates, respectively. The forget gate takes a new time step and the previous output as input and uses a sigmoid activation function to choose whether information to keep or erase. If the sigmoid function yields a value of 1, the data will be retained; if the function yields a value of 0, the data will be eliminated in its entirety. The calculation of the forget gates is shown in equation (1).

The following stage specifies what new data needs to be stored in the cell state and is divided into two pieces. The first component is the input gate, which specifies what new data is added to the cell state from the current input (Xt,ht−1). The second component is the Tanh activation function, which can be added to the cell state and produces a new candidate value $\check{C}_t$ a vector. To produce a new cell state, Ct, the multiplication of these two components will be added to the multiplication of the forget gate with the previous cell state. A portion of the data that was previously designated to be erased will be forgotten when the forget gate is multiplied by the prior cell state. Then, using its $\check{C}_t$, the new candidate values are scaled according to how much the cell state is altered. The computation of the input gate, new candidate values, and cell state are demonstrated by equations (2)-(4).

Ultimately, the output gate is calculated by employing two distinct activation functions on filtered data, while also indicating the subsequent concealed state. Initially, the sigmoid activation function receives the current input time stepx and the prior hidden state ht−1. The Tanh activation function is then supplied the revised cell state. The next hidden state is produced by multiplying the output of the sigmoid function by the output of the tanh functions. Information is passed to the following time step by the newly created hidden state and the modified cell state. The output gate and hidden state computations are displayed in equations (5) and (6).

$f_t = \sigma(W_f.[h_{t-1},x_t] + b_f)$ f_t represents forget gate. (1)

$i_t = \sigma(W_i. .[h_{t-1},x_t] + b_i)$ i_t represents input gate. (2)

$\check{C}_t = \tanh(W_C.[h_{t-1},x_t] + b_c)$ $\check{C}_t$ represents candidate values. (3)

$C_t = f_t \times C_{t-1} + i_t \times \check{C}_t$. C_t represents cell state. (4)

$o_t = \sigma(W_o.[h_{t-1},x_t] + b_o)$ o_t represents output gate. (5)

$h_t = o_t \times \tanh C_t$ h_t represnets hidden state. (6)

where W is the weight matrix, tanh is the hyperbolic tangent activation function, σ is the sigmoid activation function, and x is the input data.

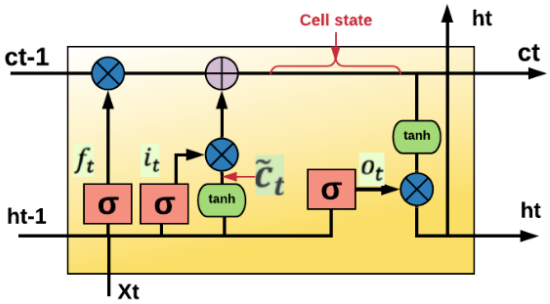


Figure 3 Single LSTM cell

LSTM has been used in activity recognition applications, with encouraging results [4], [9], [10], [11], [12].In this study, LSTM is used as a temporal model. The employed LSTM model is developed to tackle the binary classification problem effectively. The following steps outline how to develop an LSTM model for detecting human presence.

A. Data Labeling

In the data labeling process, the assignment of labels to every data point or sequence is done, marking the existence or non-existence of human activity. Label

assignment takes place in the following way as described below.

0 (Non-Human): This label indicates that no human activity was detected in the sensor data for the given time frame.

1 (Human): The label indicates that human activity was detected and may be due to single or multiple person/animal activity around the sensor.

B. Data Preparation

Once data labelling is done, we will load the labelled ultrasonic sensor data of several CSV files and be able to ask the user through the graphical interface for the file where the respective sequences for the given labels are found; the last column represents the binary target variable.

C. Data Preprocessing

The loaded data underwent several pre-processing steps.

1. Feature Extraction: Sequences (features) and labels are extracted from CSV files. Each of the final columns represents the label, and the remaining ones make up the columns that constitute the sequence data.
2. Data Reshaping: The data was reshaped into sequences, which follow the required LSTM input reshaping set by the LSTM. Reshaping is done to allow the network to learn well from the temporal structure of the data.
3. Data Splitting: The data sets was divided as follows.
   - Training Set: 64% of the data, used to train the model.
   - Validation Set: 16% of the data, used to tune the hyperparameters and prevent overfitting.
   - Test Set: 20% of the data, used to evaluate the model's performance on unseen data.

D. Building LSTM Model

The LSTM model was carefully designed to address the binary classification problem. The structure included:

1. LSTM Layers: Two LSTM layers were used, with the first returning sequences to provide a rich temporal representation for the following layers. Regularization was applied to prevent overfitting.

2. Batch Normalization Layer: Included after the first LSTM layer to ensure stability in the network's activations by normalizing the input features.

3. Dropout Layers: Deployed after LSTM and Dense layers to mitigate overfitting by randomly dropping a fraction of the input units.

4. Dense Layers: A fully connected layer followed by the LSTM layers to interpret the features, with a final output layer using a sigmoid activation function for binary classification.

The model was compiled with Adam optimizer for efficient training and binary cross-entropy as the loss function, suitable for binary classification tasks[13]. Figure 4 displays the architecture of the LSTM model.

E. Model Compilation

The Long Short-Term Memory (LSTM) model is compiled with the definition of optimizer, loss function, and evaluation metrics to guide the model's learning process and, at the same time, measure it.

Optimizer: The Adam optimizer is most welcome in applications like human presence detection for the LSTM model, which is efficient with the capability of an adaptive learning rate to aid faster convergence.

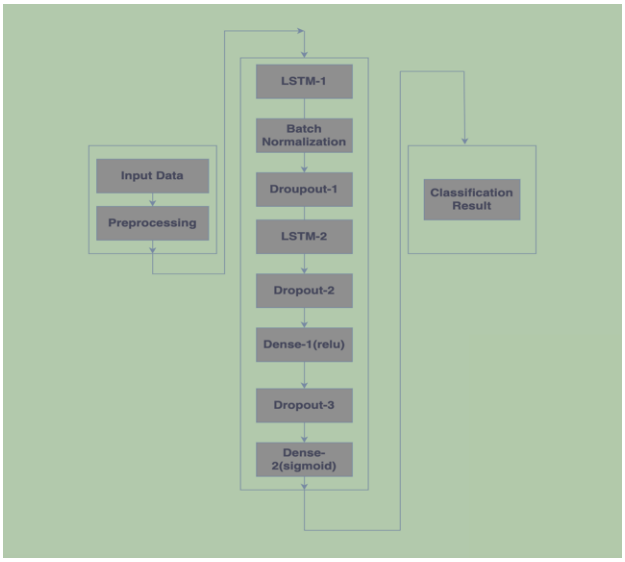


Figure 4 LSTM Classification Model

1. Loss Function: Binary cross-entropy loss effectively measures the difference between the predicted probability and actual binary labels at every point.
2. Metrics: The main metric that will be applied in evaluating model performance is "accuracy," representing the number of correct predictions from the predictions made. It gives intuitive measures of model effectiveness, where further metrics like precision, recall, and the F1 score give a more detailed explanation about predictive accuracy.

Example Compilation:
model.compile(optimizer=Adam(learning_rate=1e-3), loss='binary_crossentropy', metrics=['accuracy'])

The binary cross-entropy for the loss, an Adam optimizer, and the metric as accuracy will set up the LSTM model to learn from the training data effectively for the goal of high precision in prediction regarding the presence of human beings[14]. It is thus very important to select these individual aspects carefully, when combined together it will increase the model accuracy.

F. Model Training

The training process is designed to optimize the model's weights and biases to minimize the loss function, ultimately aiming to increase the model's accuracy in predictions. To enhance the training process, the following were included

1. Custom Call-back: It contains a custom call-back to log the training metadata, including loss and validation loss after every epoch. This feature, therefore, allows a model to monitor the training process and hence gives much insight into the performances of a model over time[15].

2. Model Checkpointing: Model Checkpoint was used to save the model at epochs where the validation loss improved[16]. This way, it was assured that the model recovers the best model seen during training and more so, what is much better with this feature is that in case of training interruption, the most recent epoch checkpointed can be used for resuming training, thus saving the progress achieved.

The model underwent training for a specified number of epochs, with early stopping applied on the validation loss to prevent overfitting. A critical phase involved retraining the model, adjusting the weights meticulously to minimize loss and enhance accuracy on unseen data.

In the final step of training, the model was saved, encapsulating all the fine-tuning and optimizations made. This ensures that the trained model, with its optimized state, can be easily deployed or used for further evaluations without the need to retrain from scratch.

G. Model Evaluation and its Metrics

After training the LSTM model for human presence detection and saving the best iteration of it using Model Checkpoint, another very important step is evaluating the model. In this phase, we load the saved model and make predictions from the test dataset, which is a set of data that the model would not have seen in the training. Hence, this is a very important step, as it must be checked that the model has actual predictive powers, and therefore the model has to generalize well on the new, unseen data.

Evaluating the performance of a model is a critical step in any machine learning task. The evaluation metrics quantify how well the model's predictions match the actual labels. For binary classification problems, such as human presence detection, commonly used metrics include accuracy, precision, recall, and the F1 score[17]. Each of these metrics offers a different perspective on the model's performance.

Accuracy: This metric measures the proportion of true results (both true positives and true negatives) among the total number of cases examined. It is a useful indicator when the classes are balanced.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision: Precision assesses the model's performance in terms of the proportion of positive identifications that were correct. It is especially important when the cost of false positives is high.

$$Precision = \frac{TP}{TP + FP}$$

Recall (Sensitivity): Recall calculates the proportion of actual positives that were identified correctly. It is crucial when the cost of false negatives is significant.

$$Recall = \frac{TP}{TP + FN}$$

F1 Score: The F1 score is the harmonic mean of precision and recall, providing a balance between the two. It is particularly useful when there is an uneven class distribution, as accuracy alone can be misleading.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Confusion Matrix A confusion matrix is a table used to describe the performance of a classification model[18]. It allows for the visualization of the model's predictions in detail, showing how the predicted classes compare to the actual labels. Here's what each term represents:

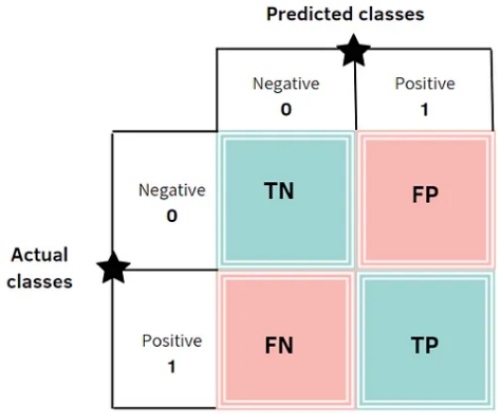


Figure 5 Confusion Matrix

True Positives (TP): These are the cases in which the model correctly predicts the positive class (e.g., correctly predicting human presence).

True Negatives (TN): These are the cases in which the model correctly predicts the negative class (e.g., correctly predicting the absence of human presence).

False Positives (FP): These occur when the model incorrectly predicts the positive class (e.g., predicting human presence when there is none).

False Negatives (FN): These occur when the model incorrectly predicts the negative class (e.g., predicting no human presence when there is).

The confusion matrix provides a comprehensive snapshot of the model's performance across all possible outcome categories. By examining the confusion matrix, one can determine not only the overall accuracy but also the types of errors the model is making. For instance, in the context of human presence detection:
A high number of FP could mean that the model is too sensitive and is picking up on irrelevant activity.
A high number of FN could indicate that the model is missing out on detecting actual human presence, which might be a serious concern in security systems.

Thus, using these metrics and the confusion matrix together provides a holistic view of the model's predictive capabilities and can guide further refinement of the model.

H. Model Optimization

Where the performance of the model is observed to drop much below the expectation, some level of refinement in the model can be done through suitable changes in hyperparameters like the learning rate, batch size, and number of epochs. In fact, the architecture of the network can be further adapted by modifying the layer structure of the LSTM itself. For example, the number of LSTM units can be increased or decreased, fully connected layers can be added or removed, and so on.

I. New Data Implementation

Once the model successfully trains and validates, this learned LSTM model will be implemented over fresh datasets to predict the based outcomes. When new time series data comes in, the model computes and renders a verdict in classifying the presence or absence of human activity based on a pattern learned from the data.

III. IMPLEMENTATION

The main steps in implementing the Long Short-Term Memory (LSTM) model for detecting human presence involve setting up the development environment and implementing machine learning models to derive the required predictions. This section offers a detailed overview of the process.

Development Environment Setup

- Software Requirements: Conda environment with necessary Python packages include TensorFlow (or Keras), NumPy, scikit-learn, and Matplotlib. These tools are essential for development tasks such as numerical computations, data manipulation, and visualization.
- Hardware: A GPU with ample memory significantly accelerates LSTM model training.

This framework implementation consists of a set of Python scripts written for parts of the machine learning pipeline: data ingestion and preprocessing, model training, evaluation, prediction over test datasets with a special script for prediction over a totally new set of unseen data. Included also are the preprocessing scripts, which convert text files into CSV format and properly label the CSV files for further processing. This modular approach leads to a clear and organized workflow, entailing the different stages required in developing a strong LSTM model for human presence detection.

A. Text to csv file

The python script consists of two main methods, browse_files and convert_to_csv, implemented to select the text files through GUI and convert the selected text files into CSV files.

```
def browse_files():
    file_paths = filedialog.askopenfilenames(title='Select TXT Files', filetypes=[('Text Files', '*.txt')])
    for file_path in file_paths:
        convert_to_csv(file_path)
```

```
def convert_to_csv(input_file_path):
    df = pd.read_csv(input_file_path, sep='\t', header=None, skiprows=1)
    df_subset = df.iloc[:, 16:]
    output_file_path = input_file_path.replace('.txt', '.csv')
    df_subset.to_csv(output_file_path, index=False, header=False)
    print(f'Conversion completed. CSV file saved as {output_file_path}')
```

B. Data Labelling

The process_csv method is a very generalized function that shall process CSV files and automatically add a label for each row based on the filename. This function is very useful for preparing datasets in tasks where each entry should be labeled as pertaining to some class.

```
def process_csv(input_file_path):
    # Determine the label based on the input file name
    if 'without_person' in input_file_path:
        label = '0'
    elif 'with_person' in input_file_path:
        label = '1'
    else:
        print(f"Warning: File name doesn't contain 'without_person' or 'with_person'. Defaulting to label '0'.")
        label = '0'

    # Generate output file path based on the input file path
    output_file_path = os.path.splitext(input_file_path)[0] + '_labeled.csv'

    with open(input_file_path, 'r') as infile, open(output_file_path, 'w', newline='') as outfile:
        csv_reader = csv.reader(infile)
        csv_writer = csv.writer(outfile)

        for row in csv_reader:
            row.append(label)
            csv_writer.writerow(row)

    print(f'Values added to the last column in {output_file_path} with label {label}')
```

C. Data Loading and Reshaping

The method load_data makes it convenient to load and pre-process time-series data from CSV files for the purpose of training a Long Short-Term Memory (LSTM) model. The function conveniently wraps the importing of datasets, extracting features and labels, and reshaping the data into LSTM input format.

```
# Function to load data from CSV files
def load_data():
    """
    Load data from CSV files selected by the user using a file dialog.

    Returns:
        Tuple: x_data (numpy.ndarray), y_data (numpy.ndarray)
    """
    y = []
    x = []

    root = tk.Tk()
    root.withdraw()  # Hide the main window

    file_paths = filedialog.askopenfilenames(title=f"Select csv data files")
    for file_path in file_paths:
        signal = np.array(np.genfromtxt(file_path, delimiter=','))
        x.append(signal[:, :-1])
        y.extend([1 if label == 1 else 0 for label in signal[:, -1]])

    root.destroy()

    x_data = np.vstack(x)
    y_data = np.vstack(y).reshape(-1, 1)

    # Reshape data for LSTM input
    sequence_length = 1
    x_data = x_data.reshape(-1, sequence_length, x_data.shape[1])

    return x_data, y_data
```

D. Data Splitting

split_data method is meant to prepare data for machine learning models like Long Short-Term Memory (LSTM) networks. The main functionalities of this function are to split a dataset systematically into distinct sets for training, validation, and testing. This ensures that training, tuning, and evaluation of the model can be done in an ideal and effective manner.

```python
# Function to split data into training, validation, and test sets
def split_data(x_data, y_data):
    x_train, x_test_temp, y_train, y_test_temp = train_test_split(x_data, y_data,
                                                  test_size=0.2, random_state=42)
    x_test, x_val, y_test, y_val = train_test_split(x_test_temp, y_test_temp,
                                              test_size=0.2, random_state=42)
    return x_train, y_train, x_val, y_val, x_test, y_test
```

E. Model Architecture

build_model method is to build and compile a Long Short-Term Memory (LSTM) model for binary classifications, e.g., human presence. It builds a sequential architecture with supporting LSTM layers among other supporting layers to effectively process time series data.

```python
# Function to build the LSTM model
def build_model(input_shape, hidden_size):

    model = Sequential()
    model.add(LSTM(hidden_size, input_shape=input_shape,
                   return_sequences=True, bias_regularizer=L1L2(0.01, 0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))
    model.add(LSTM(hidden_size, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=1e-3),
                  loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

F. Model Training and Saving

The method "train_model_and_save" orchestrates the process of training a Keras Sequential model designed for tasks like binary classification with an LSTM network. This function does both model training with given datasets and mechanisms of saving checkpoints, logging training metadata in order to improve the model development workflow.

```python
# Function to train the model and save checkpoints
def train_model_and_save(model, x_train, y_train, x_val, y_val, epochs,
                         batch_size, model_save_path, initial_epoch=0):

    checkpoint_path = model_save_path.replace('.hs', '_checkpoint.h5')
    checkpoint = ModelCheckpoint(
        checkpoint_path,
        monitor='val_loss',
        verbose=1,
        save_best_only=True,
        mode='min'
    )

    current_working_dir = os.getcwd()
    metadata_file_path = os.path.join(current_working_dir, 'trained-models',
                                      'training_metadata.txt')
    log_dir = os.path.join(current_working_dir,'trained-models',"logs", "fit",
                           datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
    tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)
    custom_callback = CustomCallback(file_path=metadata_file_path)

    history = model.fit(
        x_train,
        y_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(x_val, y_val),
        callbacks=[checkpoint, custom_callback,tensorboard_callback],
        initial_epoch=initial_epoch
    )
    model.save(model_save_path)
    return history
```

G. Model Evaluation

The function, evaluate_model_with_saved_model, dedicatedly tests the performance of the trained and saved model on the provided test dataset. The function is very important to understand the performance of the model generalization for new or unseen data.

```python
# Function to evaluate the model using saved checkpoints
def evaluate_model_with_saved_model(model_path, x_test, y_test):
    # Load the saved model
    model = load_model(model_path)
    y_pred = model.predict(x_test)
    y_pred_binary = (y_pred > 0.5).astype(int)
    # F1 Score
    f1 = f1_score(y_test, y_pred_binary)
    print(f'Test F1 Score: {f1}')
    accuracy = accuracy_score(y_test, y_pred_binary)
    print(f"Accuracy: {accuracy}")
    test_acc = (y_pred_binary == y_test).sum().item() / len(y_test)
    return y_pred_binary, test_acc
```

H. Confussion matrix plot

The method plot_confusion_matrix is used in a bid to give a diagrammatic performance of a classification model, wherein the actual labels differ from the predicted labels.

```python
# Function to plot confusion matrix
def plot_confusion_matrix(save_folder, y_true, y_pred):
    cm_image_path = os.path.join(save_folder, 'lstm_confusion_matrix.png')
    cm = confusion_matrix(y_true, y_pred)
    print('Confusion Matrix:')
    print(cm)

    # Plot and save confusion matrix as PNG
    plt.figure(figsize=(8, 6))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_true))
    disp.plot(cmap='Blues', values_format='d')
    plt.title('LSTM Confusion Matrix')
    plt.savefig(cm_image_path)
    plt.close()
    print(f'Confusion Matrix image saved at: {cm_image_path}')
```

I. Training and Validation loss

After Training an LSTM model, it is important to evaluate its performance through various metrics and visualizations. Visualizing the training and validation loss provides insights into how well the model is learning and generalizing.

```python
# Plot training and validation loss
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot training and validation accuracy
plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

IV. RESULTS AND DISCUSSION

In this study, we developed an LSTM model to differentiate human presence from ultrasonic sensor data. We split our provided dataset into three subsets: training, validation, and test datasets. For example, it was the training set the model was exposed to, and, hence, during it, it learned the temporal sequences signaling the presence of a human underneath. Validation set served as another crucial check for model fine-tuning and overfit prevention, while setting an upper limit on the number of hyperparameter tuning adjustments and model tuning. Test set was useful for the purpose of assessment, showing how the model had performed, since it gave an unbiased evaluation of the predictive capabilities the model had learned. The main hyperparameter, the batch size, was

changed, and epochs were tuned to see how they have an impact on the generalization ability of the model.

A. Model Performance Analysis

The following tables in this section will show the performance of the model with respect to different batch sizes and epoch numbers. Major metrics, including accuracy, F1 score, confusion matrices among others, will indicate a multidimensional perspective for each model. For example, the models of smaller batch size reach a plateau in F1 scores way sooner, which may be strongly indicative of quicker, though perhaps less stable, convergence. Larger batch sizes denote smoother but less sharp growth in all F1 scores and would probably be the ultimate representative for a more stable learning process.

Hypertuning Parameters	Accuracy	F1 Score	Unseendata Accuracy	Unseendata F1 Score
BatchSize=32 Epoch= 30	97.83%	98.55%	95.00%	97.44%
BatchSize=64 Epoch= 30	98.21%	98.81%	97.64%	98.63%
BatchSize=128 Epoch= 30	98.20%	98.80%	98.11%	98.91%
BatchSize=256 Epoch= 30	98.17%	98.78%	98.28%	99.01%
BatchSize=32 Epoch= 50	98.38%	98.92%	98.49%	99.13%
BatchSize=64 Epoch= 50	98.52%	99.01%	98.61%	99.20%
BatchSize=128 Epoch= 50	98.58%	99.05%	98.30%	99.02%
BatchSize=256 Epoch= 50	98.67%	99.11%	98.48%	99.12%

B. Unseen Data Performance and Impact of Data Rearrangement

An excellent exhibition of flexibility, our LSTM model trained on datasets where the last 1000 columns of the signal reordered systematically at the beginning of the signal sequences. Notwithstanding the fact that this modification was brought into the structure of the training data, the prediction was highly, unwaveringly constant in its accuracy on not-rearranged, unseen data. This robustness to variation in the input sequence shows potential for this model to apply to real-world cases where, for example, the order in which sensor data is taken would not be of consistency or predictability. This result further suggests that the model managed to learn patterns in association with human presence or absence rather than memorizing the sequence of the input data, a quite important characteristic towards its reliable deployment in different and dynamic environments.

Hypertuning Parameters	Accuracy	F1 Score	Unseendata Accuracy	Unseendata F1 Score
BatchSize=32 Epoch= 30	97.97%	98.55%	87.40%	93.27%
BatchSize=64 Epoch= 30	98.08%	98.81%	87.45%	93.30%
BatchSize=128 Epoch= 30	98.34%	98.80%	87.45%	93.30%
BatchSize=256 Epoch= 30	98.27%	98.78%	87.63%	93.39%
BatchSize=32 Epoch= 50	98.80%	98.92%	87.70%	93.42%
BatchSize=64 Epoch= 50	98.83%	99.01%	87.76%	93.45%
BatchSize=128 Epoch= 50	98.67%	99.05%	87.57%	93.36%
BatchSize=256 Epoch= 50	98.80%	99.11%	87.46%	93.31%

C. Confusion Matrix Insights

The confusion matrix from the models trained with different batches and epochs showed great classification performance, with many true positives (TP) and true negatives (TN). This results in most predictions being accurate. Few false positives (FP) and false negatives (FN) underscore the precision-recall tradeoff in this model. As can be observed from the plots below, the model has a high power to distinguish between human presence or absence, characterized by few false positives (FP) and negatives (FN).

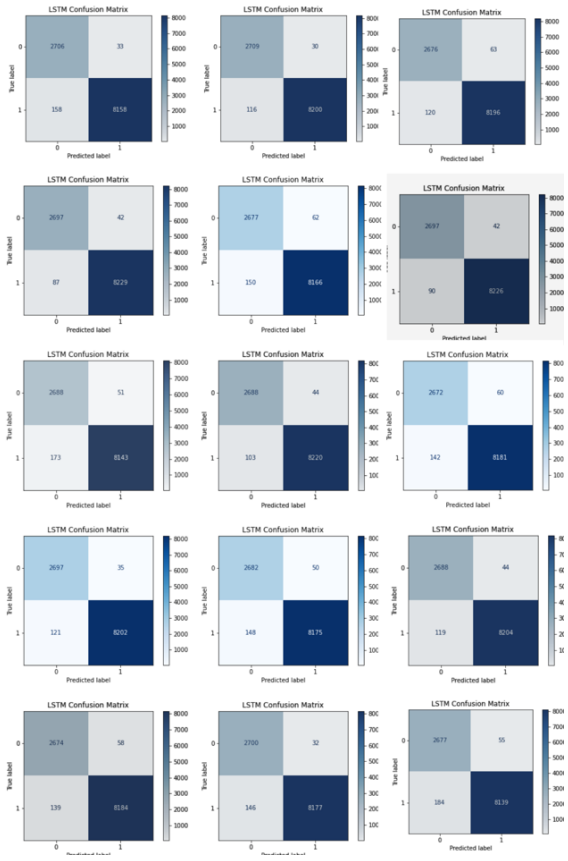
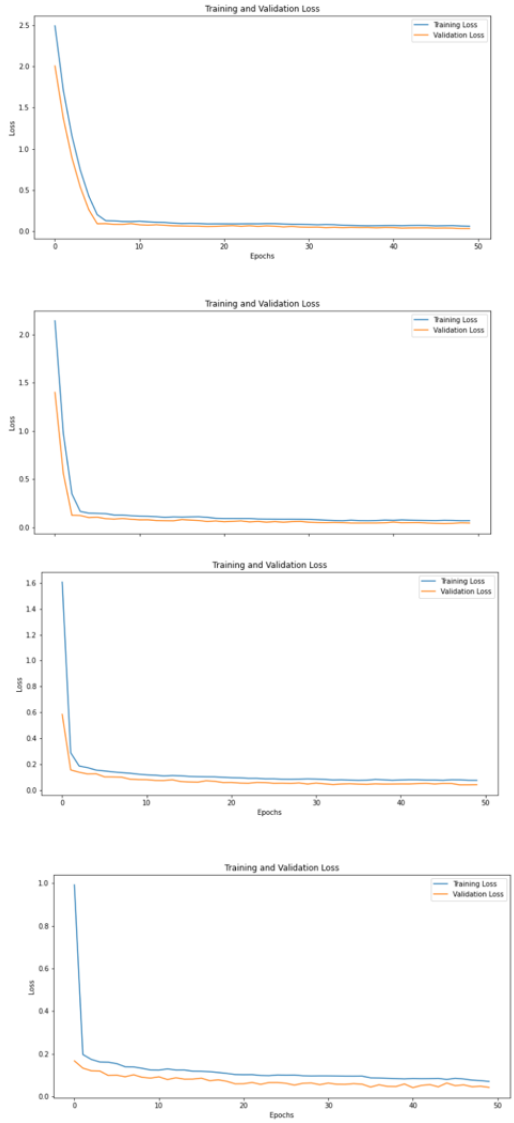


Figure 6 Confusion matrix of different models

D. Training and Validation Loss Plots

The loss plots across training and validation phases shown in **Error! Reference source not found.** indicates the model learned well without underfitting and overfitting, since the curves have downtrends that are parallel to each other. The fact that the two graphs do not diverge indicates that the model maintained its capacity for generalization



during training.

Figure 7 Training Vs Validation Loss

E. Training and Validation Accuracy Plots:

The accuracy plots in Figure 8 demonstrate a gradual increase in both training and validation accuracy over time, demonstrating that the model's classification accuracy improved as it trained. The steady increase in validation accuracy, as well as training accuracy, suggests that the model is robust and can generalize effectively to new data.

F. Implications for Batch Size:

Based on the results, larger batches appear to improve the model's performance, as seen by the higher F1 scores. This implies that the stability of gradient updates from larger batches is advantageous to the model. But there are also more computing requirements associated with this. It is possible that there is a batch size that maximizes speed while avoiding needless processing.

G. Aspects of the Epoch:

When epochs are increased, accuracy and F1 score often improve as well. This suggests that the model has effectively extracted more complex characteristics from the data with more training cycles. This should be weighed against the possibility of overfitting and the law of diminishing returns, which states that further epochs do not produce appreciable performance improvements after a certain amount of time.

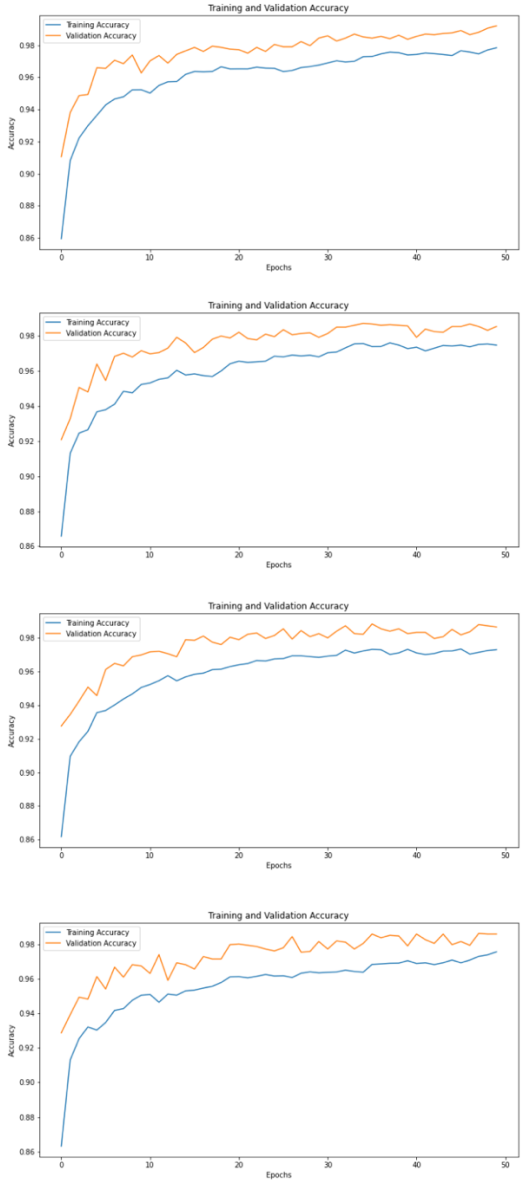


Figure 8 Training Vs Validation Accuracy

H. True Labels Vs Predicted Labels

The plots showing true versus predicted labels for a classification model. The true labels are displayed on the left, and the predicted labels are on the right.

In each row, we see that the true labels have vast periods of non-human presence (indicated by blue) and shorter periods of human presence (indicated by red). The predicted labels seem to mostly align with the true labels, with the human presence correctly identified in red.

There are also a few instances of false positives and false negatives, visible as red points among the blue for non-human periods in the predicted labels, or blue gaps within the red human periods. These
discrepancies indicate occasional errors in prediction.

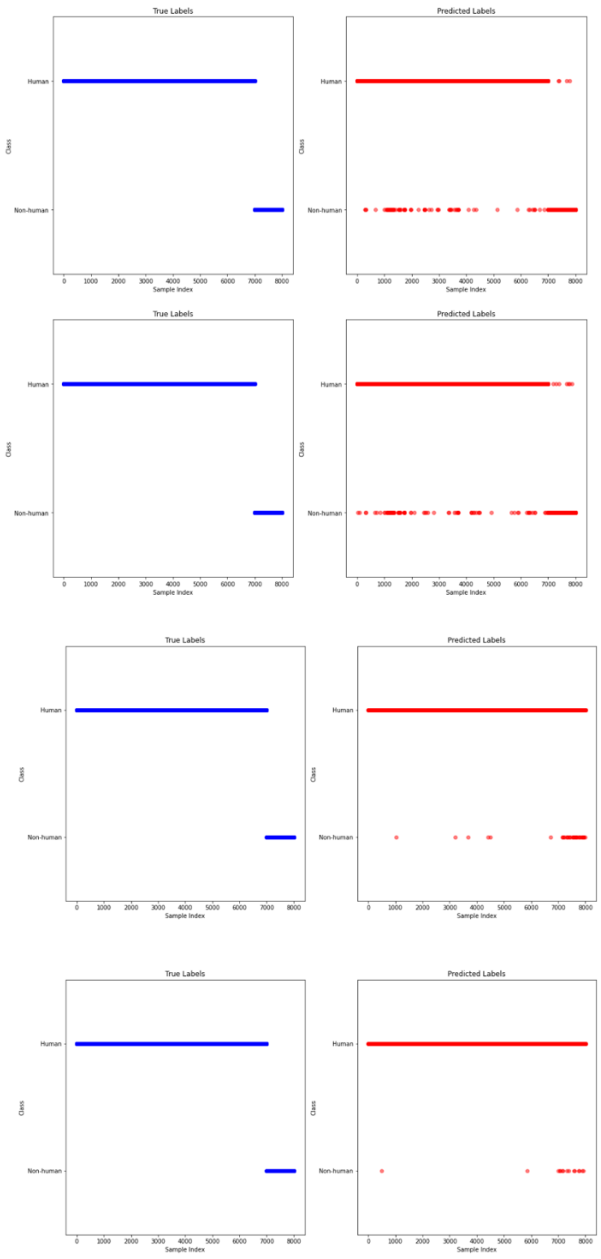


Figure 9 Prediction Comparisons

I. Model Robustness:

The performance measures indicate that the LSTM model is robust in many settings. However, the precise batch size and number of epochs can be improved further, possibly by conducting a more extensive hyperparameter search or using techniques like as cross-validation.

J. Future Directions:

Moving forward, it would be good to investigate models with different topologies or more feature engineering in order to evaluate performance metrics. Investigating the effects of data augmentation, noise reduction techniques, and more advanced regularisation methods may also result in performance improvements. Finally, the findings show that the LSTM model performs well in identifying human presence with high accuracy. Future research will try to expand on these discoveries to improve model precision, reduce computing demands, and assure scalability in a variety of operating contexts.

V. CONCLUSION

Our LSTM model, which is intended to identify human presence using time-series sensor data, has shown promise as a solid and dependable solution for a range of applications based on a thorough examination. Through a variety of batch size and epoch configurations, the model was trained to demonstrate its ability to learn from and generalise the given data.

Key findings from our study include:

1. High Model Accuracy: Across multiple batch sizes and epochs, the model maintained a high degree of accuracy, notably in distinguishing real negatives, which is critical for applications where false positives are costly.

2. Consistent performance on unseen data: The model's resilience and adaptability are demonstrated by its ability to function well on unseen data, even when input sequences are changed. This is a positive sign for deployment in real-world scenarios where data variability is a given.

3. Balanced Sensitivity and Specificity: As the batch size increased, improvements were seen in the confusion matrices, which showed a balance between sensitivity (true positive rate) and specificity (true negative rate). In situations where determining human presence is connected to safety and security protocols, this balance is essential for accuracy.

4. Model Training Dynamics: As seen by the steady decline in loss and rise in accuracy over epochs, the training and validation accuracy and loss plots verified that our model could learn efficiently over time without overfitting.

5. Computational Efficiency Considerations: While longer training times and bigger batches yielded marginal performance gains, they also came with higher computational costs. The findings imply that a compromise strategy might provide the best balance between effectiveness and efficiency.

6. Potential for Real-World Impact: The LSTM model has a great deal of potential for integration into sophisticated monitoring systems, such as sophisticated security and surveillance networks and smart home devices, given the high accuracy and robustness shown in this study.

As we proceed, it becomes evident that although our existing model works well, further optimization may be done. In order to improve the accuracy and efficiency of the model even further, future work may investigate different model architectures, put sophisticated feature engineering approaches into practice, and expand our dataset. We also understand how critical it is to continuously improve in order to stay up with changing application scenarios and real-world data trends.

Our LSTM model, in summary, is a useful tool for automated monitoring and human presence identification. Its demonstrated capacity for learning and precise prediction-making serves as evidence of the practical uses of machine learning.

REFERENCES

[1] D. Kumar and A. Sinha, "Yoga Pose Detection and Classification Using Deep Learning," *IJSRCSEIT*, pp. 160–184, Nov. 2020, doi: 10.32628/CSEIT206623.

[2] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[3] J. Medina-Quero, S. Zhang, C. Nugent, and M. Espinilla, "Ensemble classifier of long short-term memory with fuzzy temporal windows on binary sensors for activity recognition," *Expert Systems with Applications*, vol. 114, pp. 441–453, Dec. 2018, doi: 10.1016/j.eswa.2018.07.068.

[4] R. A. Hamad, A. S. Hidalgo, M.-R. Bouguelia, M. E. Estevez, and J. M. Quero, "Efficient Activity Recognition in Smart Homes Using Delayed Fuzzy Temporal Windows on Binary Sensors," *IEEE J Biomed Health Inform*, vol. 24, no. 2, pp. 387–395, Feb. 2020, doi: 10.1109/JBHI.2019.2918412.

[5] J. Collins, J. Sohl-Dickstein, and D. Sussillo, "Capacity and Trainability in Recurrent Neural Networks." arXiv, Mar. 03, 2017. doi: 10.48550/arXiv.1611.09913.

[6] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent Trends in Deep Learning Based Natural Language Processing." arXiv, Nov. 24, 2018. doi: 10.48550/arXiv.1708.02709.

[7] K. Chen, Y. Zhou, and F. Dai, "A LSTM-based method for stock returns prediction: A case study of China stock market," in *2015 IEEE International Conference on Big Data (Big Data)*, Oct. 2015, pp. 2823–2824. doi: 10.1109/BigData.2015.7364089.

[8] A. Graves, N. Jaitly, and A. Mohamed, "Hybrid speech recognition with Deep Bidirectional LSTM," in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, Dec. 2013, pp. 273–278. doi: 10.1109/ASRU.2013.6707742.

[9] N. Y. Hammerla, S. Halloran, and T. Ploetz, "Deep, Convolutional, and Recurrent Models for Human Activity Recognition using Wearables." arXiv, Apr. 29, 2016. doi: 10.48550/arXiv.1604.08880.

[10] D. Singh *et al.*, "Human Activity Recognition using Recurrent Neural Networks," vol. 10410, 2017, pp. 267–274. doi: 10.1007/978-3-319-66808-6_18.

[11] A. Murad and J.-Y. Pyun, "Deep Recurrent Neural Networks for Human Activity Recognition," *Sensors*, vol. 17, no. 11, Art. no. 11, Nov. 2017, doi: 10.3390/s17112556.

[12] R. A. Hamad, M. Kimura, and J. Lundström, "Efficacy of Imbalanced Data Handling Methods on Deep Learning for Smart Homes Environments," *SN COMPUT. SCI.*, vol. 1, no. 4, p. 204, Jul. 2020, doi: 10.1007/s42979-020-00211-1.

[13] A. Balla, M. H. Habaebi, E. A. A. Elsheikh, Md. R. Islam, F. E. M. Suliman, and S. Mubarak, "Enhanced CNN-LSTM Deep Learning for SCADA IDS Featuring Hurst Parameter Self-Similarity," *IEEE Access*, vol. 12, pp. 6100–6116, 2024, doi: 10.1109/ACCESS.2024.3350978.

[14] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." arXiv, Jan. 29, 2017. doi: 10.48550/arXiv.1412.6980.

[15] F. Chollet, *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. in Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016.

[17] S. Atawneh and H. Aljehani, "Phishing Email Detection Model Using Deep Learning," *Electronics*, vol. 12, no. 20, Art. no. 20, Jan. 2023, doi: 10.3390/electronics12204261.

[18] "Medium," Medium. Accessed: Mar. 23, 2024. [Online]. Available: https://pub.towardsai.net/deep-understanding-of-confusion-matrix-6ab1f88a267e00.00.00%2000:00:00