Configuration Manual

MSc Research Project
Data Analytics

Akhil Anthony
Student ID: x22209395

School of Computing
National College of Ireland

Supervisor: Mr.Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akhil Anthony
Student ID:	x22209395
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Mr.Hicham Rifai
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	631
Page Count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	16th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Akhil Anthony
x22209395

1 Introduction

This Configuration hand book provides complete information on software, hardware and all the steps required to develop the the Aero Engine Maintenance System.
The software and hardware requirements are specified in section 2, followed by the development of Remaining Useful Life prediction model, Defect Detection model and Web application.

2 System Configuration

The hardware and software specification are mentioned in this section.

2.1 Hardware Requirement

Table 1: Hardware Requirement

Operating System	Windows 11
Ram	8.0 GB
Hard Disc	250 GB

2.2 Software Requirement

Table 2: Software Requirement

Programming Tools	Google Colab, Visual Studio Code
Web Browser	Google Chrome or Brave
Other Required Software	Overleaf, Microsoft Word, and Microsoft EXcel

3 Remaining Useful Life (RUL)

This section consist of complete information in developing the CNN-BiLSTM model for RUL prediction. The section include Loading data, Calculating Rul, EDA, Model building and Training, Prediction and Testing

Load Data and Histogram

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns

def load_data(fleet: int):
    index_names = ['unit_number', 'time_cycles']
    setting_names = ['setting_1', 'setting_2', 'setting_3']
    sensor_names = ['s_{}'.format(i+1) for i in range(0,21)]
    col_names = index_names + setting_names + sensor_names

    train_string = f'train_FD00{fleet}.txt'
    test_string = f'test_FD00{fleet}.txt'
    RUL_string = f'RUL_FD00{fleet}.txt'

    train = pd.read_csv(train_string, sep='\s+', header=None, names=col_names)
    test = pd.read_csv(test_string, sep='\s+', header=None, names=col_names)
    y_test = pd.read_csv(RUL_string, sep='\s+', header=None, names=['RUL'])

    return train, test, y_test

# Load data for fleet 1 (or adjust fleet number as needed)
train, test, rul = load_data(fleet=1)

# Plot the distribution of time cycles in the training data
plt.figure(figsize=(10, 6))
sns.histplot(train['time_cycles'], bins=50, kde=True)
plt.title('Distribution of Time Cycles in Train')
plt.xlabel('Time Cycles')
plt.ylabel('Frequency')
plt.show()

# Merge test data with RUL for analysis
test_rul = test.copy()
test_rul['RUL'] = np.nan

for i, unit in enumerate(test_rul['unit_number'].unique()):
    max_cycle = test_rul[test_rul['unit_number'] == unit]['time_cycles'].max()
    test_rul.loc[test_rul['unit_number'] == unit, 'RUL'] = rul.iloc[i, 0] + max_cycle - test_rul['time_cycles']

# Plot the distribution of RUL in the test data
plt.figure(figsize=(10, 6))
sns.histplot(test_rul['RUL'], bins=50, kde=True)
plt.title('Distribution of RUL in Test Data')
plt.xlabel('Remaining Useful Life (RUL)')
plt.ylabel('Frequency')
plt.show()
```

Figure 1: Loading and plotting the histograms

Exploratory Data Analysis

```python
# Display the first few rows of the training data
print("Training Data:")
print(train.head())

# Display the first few rows of the test data
print("\nTest Data:")
print(test.head())

# Display the first few rows of the RUL data
print("\nRUL Data:")
print(rul.head())

# Get the shape of the datasets
print("\nShapes:")
print(f"Train: {train.shape}")
print(f"Test: {test.shape}")
print(f"RUL: {rul.shape}")

# Display data types
print("\nData Types:")
print(train.dtypes)

# Check for missing values in the training data
print("\nMissing Values in Training Data:")
print(train.isnull().sum())

# Check for missing values in the test data
print("\nMissing Values in Test Data:")
print(test.isnull().sum())

# Summary statistics for numerical features in the training data
print("\nSummary Statistics for Training Data:")
train.describe()

# Distribution of sensor readings
sensor_columns = [f's_{i}' for i in range(1, 22)]

# Plot histograms of sensor readings
train[sensor_columns].hist(bins=50, figsize=(20, 15))
plt.suptitle('Histograms of Sensor Readings')
plt.show()

import seaborn as sns
# Correlation matrix
correlation_matrix = train[sensor_columns].corr()

# Heatmap of the correlation matrix
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix of Sensor Readings')
plt.show()

train, test, rul = load_data(1)
```

Figure 2: Exploratory Data Analysis

RUL Calculation and Data Transformation

```python
# Calculate RUL for training data
def calculate_rul(df):
    max_cycle = df.groupby('unit_number')['time_cycles'].max().reset_index()
    max_cycle.columns = ['unit_number', 'max_cycle']
    df = df.merge(max_cycle, on='unit_number')
    df['RUL'] = df['max_cycle'] - df['time_cycles']
    df.drop(['max_cycle'], axis=1, inplace=True)
    return df

train = calculate_rul(train)

# Drop irrelevant columns
drop_list = ['unit_number', 'time_cycles', 'setting_1', 'setting_2', 'setting_3']
train.drop(columns=drop_list, inplace=True)

# Scale the data
scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train.drop(columns=['RUL']))
test_scaled = scaler.transform(test.drop(columns=drop_list))

X_train = train_scaled
y_train = train['RUL'].values

# Reshape data for LSTM
def reshape_data(data, sequence_length):
    X = []
    y = []
    for i in range(len(data) - sequence_length):
        X.append(data[i:i+sequence_length, :-1])
        y.append(data[i+sequence_length, -1])
    return np.array(X), np.array(y)

sequence_length = 50
X_train, y_train = reshape_data(np.hstack((train_scaled, y_train.reshape(-1, 1))), sequence_length)
```

Figure 3: Calculating RUL and Data Transformation

Model Building and Training

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Bidirectional, LSTM, Dropout, Dense

# Parameters
sequence_length = 50
num_features = 21     # Number of features (sensors)

# Build the model
model = Sequential()

# Add CNN layers
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(sequence_length, num_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))

# Add BiLSTM layers
model.add(Bidirectional(LSTM(units=50, return_sequences=True)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(units=50)))
model.add(Dropout(0.2))

# Output layer
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Print the model summary
model.summary()

model.build(input_shape=(None, sequence_length, num_features))

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=64, validation_split=0.2, verbose=1)
```

Figure 4: Model Building and model Training

Test Data and Prediction

```python
print("Drop list:", drop_list)

# Prepare the test data
test['RUL'] = rul['RUL']

# Verify and drop columns
columns_to_drop = ['RUL'] + drop_list
print("Columns to drop:", columns_to_drop)

# Drop columns if they exist in the test DataFrame
test_features = test.drop(columns=[col for col in columns_to_drop if col in test.columns])

# Scale the features
test_scaled = scaler.transform(test_features)

# Combine scaled features with the target column
test_scaled_with_rul = np.hstack((test_scaled, test['RUL'].values.reshape(-1, 1)))

# Reshape data for the model
X_test, y_test = reshape_data(test_scaled_with_rul, sequence_length)

# Predict
y_pred = model.predict(X_test)
```

Figure 5: Preparing Test Data and Prediction

MAE and MSE

```python
# Remove samples where y_test is NaN
valid_indices = ~np.isnan(y_test)
X_test = X_test[valid_indices]
y_test = y_test[valid_indices]
y_pred = y_pred[valid_indices]

# Create the metric objects
mae_metric = tf.keras.metrics.MeanAbsoluteError()
mse_metric = tf.keras.metrics.MeanSquaredError()

# Update the state of the metrics with the true values and predictions
mae_metric.update_state(y_test, y_pred.flatten())
mse_metric.update_state(y_test, y_pred.flatten())

# Get the results
mae = mae_metric.result().numpy()
mse = mse_metric.result().numpy()

print(f'MAE: {mae}, MSE: {mse}')

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.title('Mean Absolute Error')

plt.show()
```

Figure 6: MAE and MSE

Testing

```python
# Export the model
model.save('bilstm_model.h5')

import numpy as np

def predict_rul(sensor_data, sequence_length=50):
    # Print length of sensor_data for debugging
    print(f"Length of sensor_data: {len(sensor_data)}")

    # Check if sensor_data length matches the number of features
    if len(sensor_data) != 21:
        raise ValueError("The input sensor_data must contain exactly 21 sensor readings.")

    # Create a sequence with the provided sensor data
    # Assume the rest of the sequence is filled with zeroes
    full_sequence = np.zeros((sequence_length, 21))
    full_sequence[-1] = sensor_data  # Place the provided sensor data at the end of the sequence

    # Reshape for model input
    sensor_data = full_sequence.reshape(1, sequence_length, 21)  # (batch_size, sequence_length, num_features)

    # Make predictions
    prediction = model.predict(sensor_data)

    return prediction[0][0]

# Example sensor readings with exactly 21 values
sensor_readings = [
    106.82, 107.12, 106.92, 107.05, 106.88, 107.10,
    106.95, 107.00, 107.15, 106.90, 107.05, 106.85,
    107.12, 106.88, 107.00, 107.10, 106.92, 107.20,
    107.00, 106.95, 107.05  # Ensure this list has 21 elements
]

# Get the RUL prediction
rul_prediction = predict_rul(sensor_readings)
print(f"Predicted Remaining Useful Life (RUL): {rul_prediction}")
```

Figure 7: Testing

4 Defect Detection

This section consist of complete information in developing the VGG16-SVM model for Defect Detection. The images include Libraries, Loading the data, Feature Extraction, Transformation, Merging the label and Feature data, Model building and Training, Prediction, and Testing.

Import Libraries

```
import base64
import pandas as pd
from arrow import now
from glob import glob
from io import BytesIO
from os.path import basename
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score

import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten

from umap import UMAP
```

Figure 8: Libraries

Data Preprocessing

```
#load data
GLOB = '/content/drive/MyDrive/Aero-engine_defect-detect_new/images'
SIZE = 512
STOP = 500
THUMBNAIL_SIZE = (100, 100)

# Featur extraction vgg16 model
def embed_vgg16(model, filename: str):
    with Image.open(fp=filename, mode='r') as image:
        image = image.resize((224, 224))
        image = np.array(image)
        image = image / 255.0  # Normalize
        image = np.expand_dims(image, axis=0)
        return model.predict(image).flatten()

# flatten
def flatten(arg):
    return [x for xs in arg for x in xs]

# base64 convertion
def png(filename: str) -> str:
    with Image.open(fp=filename, mode='r') as image:
        buffer = BytesIO()
        image.resize(size=THUMBNAIL_SIZE).save(buffer, format='png')
        return 'data:image/png;base64,' + base64.b64encode(buffer.getvalue()).decode()

# process image from directory
def get_picture_from_glob(arg: str, tag: str, stop: int) -> list:
    time_get = now()
    result = [pd.Series(data=[tag, basename(input_file), embed_vgg16(model=model, filename=input_file), png(filename=input_file)],
    index=['tag', 'name', 'value', 'image'])
        for index, input_file in enumerate(glob(pathname=arg)) if index < stop]
    print('encoded {} data {} rows in {}'.format(tag, len(result), now() - time_get))
    return result

# vgg16 feature extraction
vgg16_base = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = vgg16_base.output
x = Flatten()(x)
model = Model(inputs=vgg16_base.input, outputs=x)

# embeding and merging data
time_start = now()
data_dict = {basename(folder) : folder + '/*.jpg' for folder in glob(GLOB + '/*')}
df = pd.DataFrame(data=flatten(arg=[get_picture_from_glob(arg=value, tag=key, stop=STOP) for key, value in data_dict.items()]))
df['label name'] = df['name'].apply(func=lambda x: x.replace('.jpg', '.txt'))
print('done in {}'.format(now() - time_start))
```

Figure 9: Loading data, Feature Extraction, Transformation, and Merging Data

Merging Data for VGG16-SVM model

```
# label data load and merge
from os.path import basename, join
root = '/content/drive/MyDrive/Aero-engine_defect-detect_new/labels/'
dfs = []
for subfolder in ['val', 'train']:
    for input_file in glob(join(root, subfolder, '*.txt')):
        current_df = pd.read_csv(input_file, sep=' ', names=['label', 'w0', 'w1', 'w2', 'w3'])
        current_df['tag'] = [subfolder] * len(current_df)
        current_df['name'] = basename(input_file)
        dfs.append(current_df)

labels_df = pd.concat(dfs, axis=0)

# Merge dataframes
all_df = df.merge(labels_df, left_on='label name', right_on='name', how='left')

# merged data
all_df['class'] = all_df['label'].map({0: 'scratch', 1: 'dot', 2: 'crease', 3: 'damage'})
all_df.head()

# validation and train data
val_df = all_df[(all_df['tag_x'] == 'val') & (all_df['tag_y'] == 'val')]
train_df = all_df[(all_df['tag_x'] == 'train') & (all_df['tag_y'] == 'train')]

train_df.head()

val_df.head()

# histogram
sns.histplot(data=train_df, x='class')
plt.title('Class Distribution in Training Data')
plt.show()
```

Figure 10: Loading the Label data, Merging With the Extracted Data and Histogram

UMAP

```
# from umap import UMAP
from datetime import datetime
time_start = datetime.now()
umap = UMAP(random_state=2024, verbose=True, n_jobs=1, low_memory=False, n_epochs=2000)
all_df[['x', 'y']] = umap.fit_transform(X=all_df['value'].apply(pd.Series))

time_end = datetime.now()
print(f"UMAP fitting completed. Time taken: {time_end - time_start}")

from bokeh.models import ColumnDataSource, HoverTool
from bokeh.palettes import Category20_4
from bokeh.plotting import figure, output_notebook, show
from bokeh.transform import factor_cmap

output_notebook()

plot_df = all_df.dropna(subset=['class'])[['x', 'y', 'class', 'image']]
datasource = ColumnDataSource(plot_df)
mapper = factor_cmap('class', palette=Category20_4, factors=['damage', 'crease', 'scratch', 'dot'])

plot_figure = figure(title='UMAP projection: engine defects', width=1000, height=800, tools=('pan, wheel_zoom, reset'))

plot_figure.add_tools(HoverTool(tooltips="""
<div>
    <div>
        <img src='@image' style='float: left; margin: 5px 5px 5px 5px'/>
    </div>
    <div>
        <span style='font-size: 16px; color: #224499'>Class:</span>
        <span style='font-size: 18px'>@class</span>
    </div>
</div>
"""))

plot_figure.circle('x', 'y', source=datasource, color=mapper, line_alpha=0.6, fill_alpha=0.6, size=5,)
show(plot_figure)
```

Figure 11: UMAP and Result Visualization

```python
# Separate features and target
X_train = np.vstack(train_df['value'])
y_train = train_df['class']
X_val = np.vstack(val_df['value'])
y_val = val_df['class']

# Initialize and fit the scaler
scaler = StandardScaler()
train_features = scaler.fit_transform(X_train)
val_features = scaler.transform(X_val)

import joblib  # Import joblib for saving the model and scaler
```

Figure 12: Data Transformation

```python
# Initialize and train the SVM model
svm = SVC(kernel='linear', C=1, probability=True)
svm.fit(train_features, y_train)

# Save the scaler and the trained SVM model
joblib.dump(scaler, 'scaler.pkl')
# joblib.dump(svm, 'svm_model.pkl')

# Save the scaler and the trained SVM model
# joblib.dump(scaler, 'scaler.pkl')
joblib.dump(svm, 'svm_model.pkl')

val_predictions = svm.predict(val_features)

print('SVM accuracy: {:5.4f}'.format(accuracy_score(y_true=val_df['class'], y_pred=val_predictions)))

print(classification_report(y_true=val_df['class'], y_pred=val_predictions))
```

Figure 13: Classification Matrix

Preprocessing Test Data

```python
import numpy as np
from PIL import Image
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten
import joblib  # For loading the saved scaler and SVM model

# Function to load and preprocess the image
def load_and_preprocess_image(image_path):
    with Image.open(image_path) as img:
        img = img.resize((224, 224))
        img_array = np.array(img) / 255.0  # Normalize the image
        img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array

# Function to extract features using VGG16
def extract_features_vgg16(model, img_array):
    return model.predict(img_array).flatten()
```

Figure 14: Loading and Feature Extraction of Test Data

Prediction

```python
def predict_image_class(image_path, model, svm, scaler, class_mapping, threshold=0.7):
    img_array = load_and_preprocess_image(image_path)
    features = extract_features_vgg16(model, img_array)
    scaled_features = scaler.transform([features])
    confidence_scores = svm.predict_proba(scaled_features)[0]
    prediction = svm.classes_[np.argmax(confidence_scores)]
    max_confidence = np.max(confidence_scores)

    print(f'Prediction: {prediction}')  # Print the prediction value
    print(f'Confidence Scores: {confidence_scores}')  # Print the confidence scores
    print(f'Max Confidence: {max_confidence}')  # Print the max confidence score

    if max_confidence < threshold:
        return "Unknown class", confidence_scores
    return class_mapping.get(prediction, "Unknown class")

# Load the VGG16 model for feature extraction
vgg16_base = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = vgg16_base.output
x = Flatten()(x)
vgg16_model = Model(inputs=vgg16_base.input, outputs=x)

# Load the trained SVM model and scaler
svm = joblib.load('svm_model.pkl')  # Path to your saved SVM model
# scaler = joblib.load('scaler.pkl')  # Path to your saved scaler

scaler = joblib.load('scaler.pkl')  # Path to your saved scaler

# Class mapping for predictions
class_mapping = {
    'scratch': 'scratch',
    'dot': 'dot',
    'crease': 'crease',
    'damage': 'damage'
}
```

Figure 15: Prediction Function and Saving the model

Testing

```python
# Predict the class of the image
predicted_class = predict_image_class(image_path, vgg16_model, svm, scaler, class_mapping)
print(f'The predicted class of the image is: {predicted_class}')

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_true=val_df['class'], y_pred=val_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=svm.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```

Figure 16: Testing the Model and Confusion matrix

5 Web Application

This section provides the overall information of building the Web Application using Streamlit. The images include the Defect Detection and Remaining Useful life (RUL) prediction section and the code for corresponding Buttons and Sliders.
Initially the saved models, the requirements and the code for the web application should be stored in one folder. For implementing the application virtual environment should be installed.

The steps for running the code for application are:

- pip install virtualenv
- virtualenv venv
- pip install -r requirements.txt
- venv\Scripts\activate
- streamlit application.py

Libraries

```python
import streamlit as st
import numpy as np
from PIL import Image
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten
import joblib
import tensorflow as tf


st.set_page_config(page_title="AeroInsight", page_icon="🛠️")
```

Figure 17: Libraries for Developing the Web Application

Defect Detection

```python
# Function to load and preprocess the image
def load_and_preprocess_image(image):
    img = image.resize((224, 224))
    img_array = np.array(img) / 255.0  # Normalize the image
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array

# Function to extract features using VGG16
def extract_features_vgg16(model, img_array):
    return model.predict(img_array).flatten()

def predict_image_class(image, model, svm, scaler, class_mapping, threshold=0.7):
    img_array = load_and_preprocess_image(image)
    features = extract_features_vgg16(model, img_array)
    scaled_features = scaler.transform([features])
    confidence_scores = svm.predict_proba(scaled_features)[0]
    prediction = svm.classes_[np.argmax(confidence_scores)]
    max_confidence = np.max(confidence_scores)

    if max_confidence < threshold:
        return "Unknown class", confidence_scores
    return class_mapping.get(prediction, "Unknown class"), confidence_scores

# Load the VGG16 model for feature extraction
vgg16_base = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = vgg16_base.output
x = Flatten()(x)
vgg16_model = Model(inputs=vgg16_base.input, outputs=x)

# Load the trained SVM model and scaler
svm = joblib.load('svm_model.pkl')  # Path to your saved SVM model
scaler = joblib.load('scaler.pkl')  # Path to your saved scaler

# Class mapping for predictions
class_mapping = {
    'scratch': 'scratch',
    'dot': 'dot',
    'crease': 'crease',
    'damage': 'damage'
}
```

Figure 18: Defect Detection section in Web Application

Remaining Useful Life (RUL)

```python
# Load the BiLSTM model for RUL prediction
rul_model = tf.keras.models.load_model('bilstm_model.h5')

# Function to predict RUL
def predict_rul(sensor_data, sequence_length=50):
    if len(sensor_data) != 21:
        raise ValueError("The input sensor_data must contain exactly 21 sensor readings.")

    full_sequence = np.zeros((sequence_length, 21))
    full_sequence[-1] = sensor_data  # Place the provided sensor data at the end of the sequence
    sensor_data = full_sequence.reshape(1, sequence_length, 21)  # Reshape for model input
    prediction = rul_model.predict(sensor_data)

    return prediction[0][0]

# Default values for the sliders
default_values = [
    106.82, 107.12, 106.92, 107.05, 106.88, 107.10,
    106.95, 107.00, 107.15, 80.90, 107.05, 106.85,
    107.12, 106.88, 107.00, 107.10, 106.92, 107.20,
    107.00, 106.95, 107.05
]
```

Figure 19: RUL section in Web Application

Drop down Menu adn Buttons

```python
# Streamlit app
st.title("Aero-Engine Defect Detection and RUL Analysis")

# Sidebar for navigation
option = st.sidebar.selectbox("Select Option", ("Defect Detection", "RUL Analysis"))

if option == "Defect Detection":
    st.header("Defect Detection")
    uploaded_file = st.file_uploader("Choose an image...", type=["jpg", "jpeg", "png"])

    if uploaded_file is not None:
        image = Image.open(uploaded_file)
        st.image(image, caption='Uploaded Image.', use_column_width=True)
        st.write("")

        if st.button('Classify Image'):
            st.write("Classifying...")
            predicted_class, confidence_scores = predict_image_class(image, vgg16_model, svm, scaler, class_mapping)
            st.write(f'The predicted class of the image is: {predicted_class}')
```

Figure 20: Drop down menu and Buttons to initiate Defect Detection

Sliders for Sensors

```python
else:
    st.header("Remaining Useful Life (RUL) Analysis")

    # Create sliders for each sensor with updated maximum values and default values
    sensor1 = st.slider("Sensor 1 (Fan inlet temperature)", min_value=0.0, max_value=620.0, value=default_values[0], step=0.01)
    sensor2 = st.slider("Sensor 2 (LPC outlet temperature)", min_value=0.0, max_value=750.0, value=default_values[1], step=0.01)
    sensor3 = st.slider("Sensor 3 (HPC outlet temperature)", min_value=0.0, max_value=1700.0, value=default_values[2], step=0.01)
    sensor4 = st.slider("Sensor 4 (Fan inlet temperature)", min_value=0.0, max_value=1550.0, value=default_values[3], step=0.01)
    sensor5 = st.slider("Sensor 5 (Fan inlet temperature)", min_value=0.0, max_value=120.0, value=default_values[4], step=0.01)
    sensor6 = st.slider("Sensor 6 (Fan inlet temperature)", min_value=0.0, max_value=130.0, value=default_values[5], step=0.01)
    sensor7 = st.slider("Sensor 7 (Fan inlet temperature)", min_value=0.0, max_value=660.0, value=default_values[6], step=0.01)
    sensor8 = st.slider("Sensor 8 (Fan inlet temperature)", min_value=0.0, max_value=2500.0, value=default_values[7], step=0.01)
    sensor9 = st.slider("Sensor 9 (Fan inlet temperature)", min_value=0.0, max_value=9200.0, value=default_values[8], step=0.01)
    sensor10 = st.slider("Sensor 10 (Fan inlet temperature)", min_value=0.0, max_value=110.0, value=default_values[9], step=0.01)
    sensor11 = st.slider("Sensor 11 (Fan inlet temperature)", min_value=0.0, max_value=150.0, value=default_values[10], step=0.01)
    sensor12 = st.slider("Sensor 12 (Fan inlet temperature)", min_value=0.0, max_value=630.0, value=default_values[11], step=0.01)
    sensor13 = st.slider("Sensor 13 (Fan inlet temperature)", min_value=0.0, max_value=2500.0, value=default_values[12], step=0.01)
    sensor14 = st.slider("Sensor 14 (Fan inlet temperature)", min_value=0.0, max_value=8300.0, value=default_values[13], step=0.01)
    sensor15 = st.slider("Sensor 15 (Fan inlet temperature)", min_value=0.0, max_value=110.0, value=default_values[14], step=0.01)
    sensor16 = st.slider("Sensor 16 (Fan inlet temperature)", min_value=0.0, max_value=110.0, value=default_values[15], step=0.01)
    sensor17 = st.slider("Sensor 17 (Fan inlet temperature)", min_value=0.0, max_value=500.0, value=default_values[16], step=0.01)
    sensor18 = st.slider("Sensor 18 (Fan inlet temperature)", min_value=0.0, max_value=2500.0, value=default_values[17], step=0.01)
    sensor19 = st.slider("Sensor 19 (Fan inlet temperature)", min_value=0.0, max_value=210.0, value=default_values[18], step=0.01)
    sensor20 = st.slider("Sensor 20 (Fan inlet temperature)", min_value=0.0, max_value=140.0, value=default_values[19], step=0.01)
    sensor21 = st.slider("Sensor 21 (Fan inlet temperature)", min_value=0.0, max_value=130.0, value=default_values[20], step=0.01)

    # Collect sensor data
    sensor_data = [
        sensor1, sensor2, sensor3, sensor4, sensor5, sensor6, sensor7, sensor8,
        sensor9, sensor10, sensor11, sensor12, sensor13, sensor14, sensor15,
        sensor16, sensor17, sensor18, sensor19, sensor20, sensor21
    ]

    st.write(f"Sensor data: {sensor_data}")

    # Predict RUL when the button is pressed
    if st.button("Predict RUL"):
        prediction = predict_rul(sensor_data)
        st.write(f"Predicted Remaining Useful Life (RUL): {prediction:.2f}")
```

Figure 21: Sliders for Sensors and Button to initiate REUL Prediction

6 Results

This section provides the output generated in the Aero Engine Maintenance System Web Application. The out put for both Remaining Useful Life and Defect detection is included.

Aero Engine Maintenance System

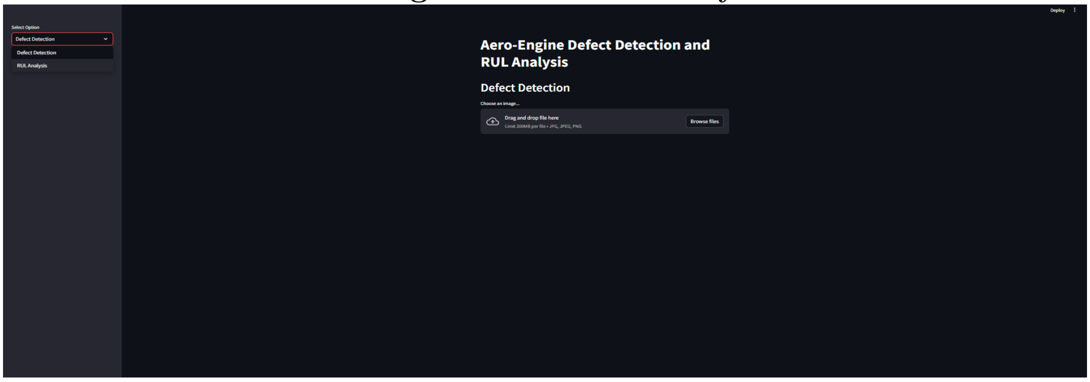

Figure 22: Interface

Defect Detection

Figure 23: Defect Detection Result

7 How to run the code?

- Rul prediction:

  Upload the train_FD001, test_FD001, and RUL_FD001 data to google colab. By clicking the run all option, the code will run automatically.

Sensor Data

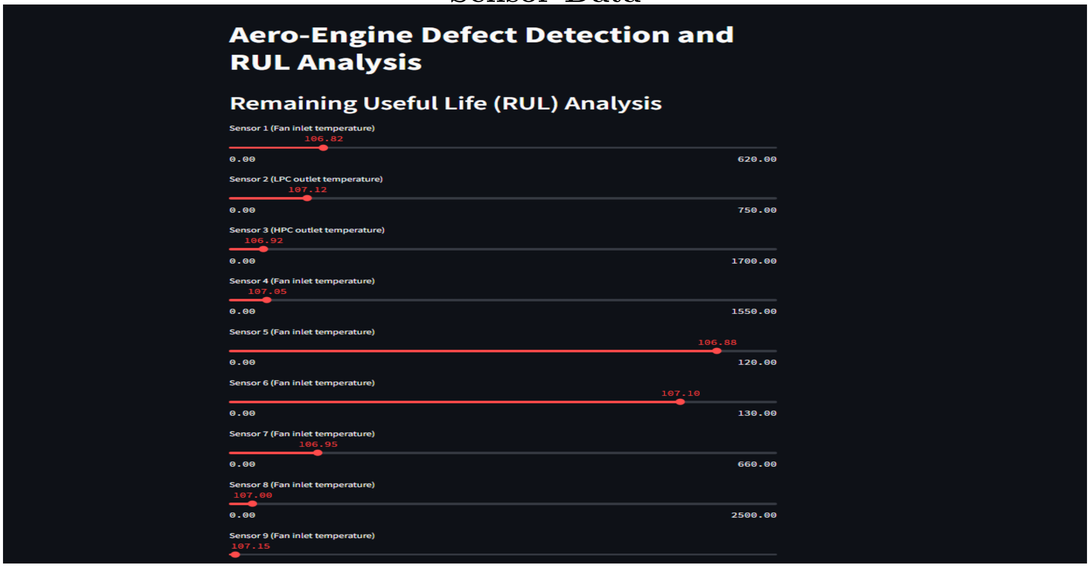

Figure 24: RUL Prediction

Sensor Data and Prediction

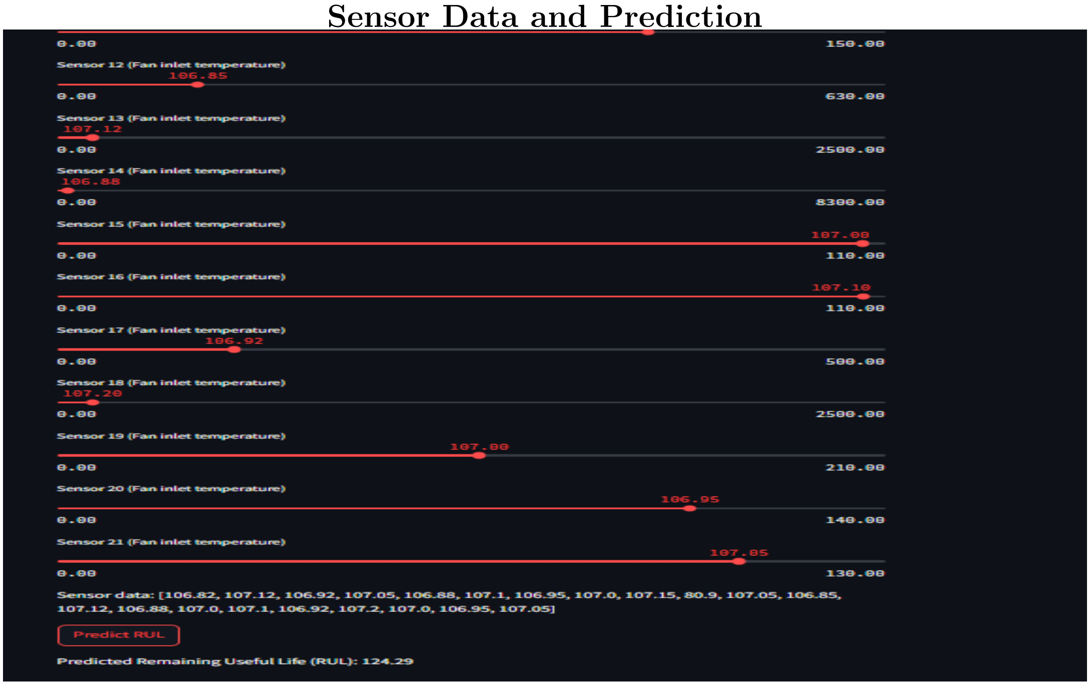

Figure 25: RUL Prediction

- Defect Detection:

  Upload the data from kaggle to google drive and mount it to google colab. The code can be run by changing the path of the input data in the code.

- Application

  Open the application folder in the visual studio code. Select the application.py pythone file, and in the terminal, select cmd and folow the steps mentioned in the application section of configuration manual.