"**Implementation coding of:** Demystifying Diagnosis: An efficient Deep Learning Technique with Explainable AI to Improve Breast Cancer Detection".

Ahmed Alzahrani[1], Muhammad Ali Raza[2], Muhammad Zubair Asghar[2].

[1]Department of Computer Science, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah 21589, Saudi Arabia.

[2]Gomal Research Institute of Computing (GRIC), Faculty of Computing, Gomal University, D.I.Khan(KP), Pakistan.

Proposed BILSTM+CNN with Explainable AI (SHAP):

PYTHON is used in the identification of breast cancer. It is an open-source data science platform called ANACONDA. Both tools are useful for applying various deep-learning models to a batch of breast cancer data.

The following provides an overview of code implementation:

Import libraries: The first step in preparing a dataset and in training a model is to load libraries. As shown in fig.1, these libraries are required to carry out a specific task.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from keras.models import Sequential
        from sklearn.preprocessing import LabelEncoder
        from keras.layers import Dense,Flatten
        from keras.layers import LSTM,BatchNormalization
        from keras.layers import Dropout
        #from keras.utils import np_utils
        from tensorflow.keras.optimizers import Adam
        #from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
        from tensorflow.keras.layers import Conv1D, MaxPool1D

In [2]: from sklearn import datasets, metrics
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
```

Fig 1. Importing libraries

The overviews of a few basic python PACKAGES are as follows:

- *Numpy:* It is fundamentally a numerical Python library, an essential module that does scientific computing tasks and calculations.
- *Pandas:* its a free, open-source Python package for analyzing data change.
- *Sklearn:* Cross fold validation, feature-extraction, and unsupervised machine/deep learning techniques are all supported by SciKit_learn, which also supports a variety of data sources.

- *Keras:* A Python library that is used for analyzing data, creating models, and displaying graphs.

Dataset uploading from computer

The dataset in "csv" form is uploaded by executing *"data = pd.read_csv"* code.

```
data = pd.read_csv('Desktop\data.csv')
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...

5 rows × 32 columns

Fig 2. Data set uploading

Dependent and independent variable

```
x = data.drop(['id', 'diagnosis'], axis='columns')
Y = data['diagnosis']
```

Fig 3. Dependent and independent variable

Feature selection

The Extra Tree Classifier feature selection method was employed in this investigation, shown in fig.4. Extremely Randomised Trees Classifier, also known as Extra Trees Classifier.
Fig 5. Demonstrate dependent and independent variable. Thirteen independent features are drop by selecting most important features using feature selection technique.

```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model=ExtraTreesClassifier()
model.fit(x,Y)
```

```
ExtraTreesClassifier()
```

```
ranked_features=pd.Series(model.feature_importances_,index=X.columns)
ranked_features.nlargest(18).plot(kind='barh')
plt.show()
```

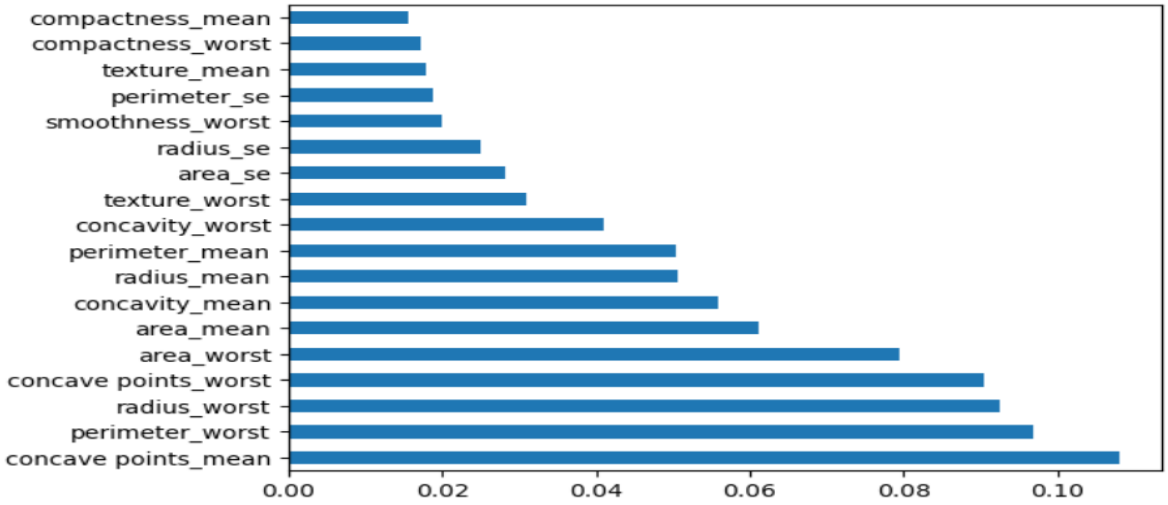


Fig 4. Feature selection code

```
X = data.drop(['id', 'diagnosis','smoothness_mean','symmetry_mean', 'fractal_dimension_mean','texture_se','smoothness_se',
'compactness_se', 'concavity_se', 'concave points_se','symmetry_se','fractal_dimension_se','symmetry_worst',
              'fractal_dimension_worst'], axis='columns')
y = data['diagnosis']
```

Fig 5. Dependent and independent variable

Imbalance Data set

The underlying dataset unfairly affects both groups due to its extreme imbalance. For a positive test, for Benign, there are 357 instances (62.7%), and for Malignant, there are 212 cases (37.2%). After applying random oversampling, the balanced data set is treated in the same way for both the M:357 and B:357 classes. There are 714 total cases, as shown in fig.6.

```
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
ros= RandomOverSampler()
X,y=ros.fit_resample(X,y)
print(Counter(y))
```

```
Counter({'M': 357, 'B': 357})
```

Fig 6. Imbalance Data set

Label Encoding

The feature "diagnosis," which belongs to the class of characters, is one of the dataset's characteristics. Using Label Encoder of the scikit learn library, this column is converted into a numeric type as seen in fig.7.

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder
labelencoder_X_1 = LabelEncoder()
y = labelencoder_X_1.fit_transform(y)
```

Fig 7. Label Encoder code

Data set train- test split

For train-test split the SciKit-learn selection model that split the dataset into two subsets:

1. Training set (model is trained using training set)
2. Testing set (model is evaluated using testing set)

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =61)
```

Fig 8. Train-Test Split

Standardization

For many ML and DL models, ensuring that the data is scaled properly is one of the fundamental functions of feature engineering. Standardization scaling technique used as shown in fig. 9.

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Fig 9. Feature scaling code

Reshape test and train set

Reshape() is a built-in function that is used to maintain consistency and meet input requirements as shown in fig. 10.

```
X_train = X_train.reshape(571,18,1)
X_test = X_test.reshape(143, 18, 1)
```

Fig 10. Reshape function.

Proposed BiLSTM-CNN construction

The proposed DL model, BiLSTM-CNN, was developed using the Keras sequential API. As seen in (Fig. 11), the model is built up layer by layer.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Bidirectional, LSTM, Dropout, Conv1D, MaxPooling1D, Flatten, Dense, Input

model = Sequential()
# Use Input() for the first layer
model.add(Input(shape=(X_train.shape[1], 1)))
model.add(Bidirectional(LSTM(units=35, activation="relu", return_sequences=True)))
model.add(Dropout(0.3))

model.add(Conv1D(filters=32, kernel_size=6, padding='same', activation='relu'))
model.add(Dropout(0.3))

model.add(MaxPooling1D(pool_size=2))

model.add(Flatten())
model.add(Dense(35, activation='relu'))

model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

Fig 11. Proposed BiLSTM-CNN construction.

Model fitting: The model for a given dataset is trained by "model.fit ()" and is used to train our model on the given dataset, as shown in Fig 12.

Batch size- Number of iteration training set samples used in one iteration. We selected 32 batch-size.

Epochs- number of iterations over training and testing sets.

Validation-data-Data that are analysed to determine the model metrics and loss at the end of each epoch.

Verbose- integer values (e.g., 0,1,2).

```
history = model.fit(X_train, y_train, epochs=100,batch_size=32 , validation_data=(X_test, y_test), verbose=1)
```

Fig 12. Proposed model fitting.

Training output: The proposed BiLSTM-CNN is trained on "571" training samples & is validated on "143" testing samples as shown in Fig 13. Loss, accuracy, validation-accuracy, and validation-accuracy are outputs of the training model.

```
Epoch 1/100
18/18 [==============================] - 5s 60ms/step - loss: 0.4944 - accuracy: 0.8897 - val_loss: 0.2435 - val_accurac
y: 0.9371
Epoch 2/100
18/18 [==============================] - 0s 23ms/step - loss: 0.1545 - accuracy: 0.9422 - val_loss: 0.1310 - val_accurac
y: 0.9441
Epoch 3/100
18/18 [==============================] - 0s 22ms/step - loss: 0.1403 - accuracy: 0.9422 - val_loss: 0.1492 - val_accurac
y: 0.9161
Epoch 4/100
18/18 [==============================] - 0s 25ms/step - loss: 0.1122 - accuracy: 0.9562 - val_loss: 0.1046 - val_accurac
y: 0.9720
Epoch 5/100
18/18 [==============================] - 0s 22ms/step - loss: 0.1007 - accuracy: 0.9632 - val_loss: 0.1055 - val_accurac
y: 0.9790
Epoch 6/100
18/18 [==============================] - 0s 21ms/step - loss: 0.1051 - accuracy: 0.9580 - val_loss: 0.1082 - val_accurac
y: 0.9650
Epoch 7/100
```

Fig 13. Training model outputs

Model summary: Summary of proposed BiLSTM-CNN model is shown in fig.14.

```
print(model.summary())
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona  (None, 18, 70)           10360
 l)

 dropout (Dropout)           (None, 18, 70)            0

 conv1d (Conv1D)             (None, 18, 32)            13472

 dropout_1 (Dropout)         (None, 18, 32)            0

 max_pooling1d (MaxPooling1D  (None, 9, 32)            0
 )

 flatten (Flatten)           (None, 288)               0

 dense (Dense)               (None, 35)                10115

 dense_1 (Dense)             (None, 1)                 36

=================================================================
Total params: 33,983
Trainable params: 33,983
Non-trainable params: 0
_________________________________________________________________
None
```

Fig 14. Model Summary.

Model Evaluation: The BiLSTM-CNN model is evaluated through "model.evaluate()" function as seen in Fig 15.

```
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

```
Test loss: 0.04880256950855255
Test accuracy: 0.9930070042610168
```

Fig 15. Model evaluation.

Prediction: For projection and predication the dataset “Prediction()” function as shown in Fig 16.

```
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
from sklearn import metrics
print(metrics.accuracy_score(y_test,y_pred))
```

```
0.993006993006993
```

Fig 16. Model prediction.

confusion matrix: The confusion matrix is obtained using the "confusion_matrix()" function and is shown in Fig 17.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[61,  1],
       [ 0, 81]], dtype=int64)
```

```
import seaborn as sns
sns.heatmap(cm,annot=True)
plt.savefig('h.png')
```

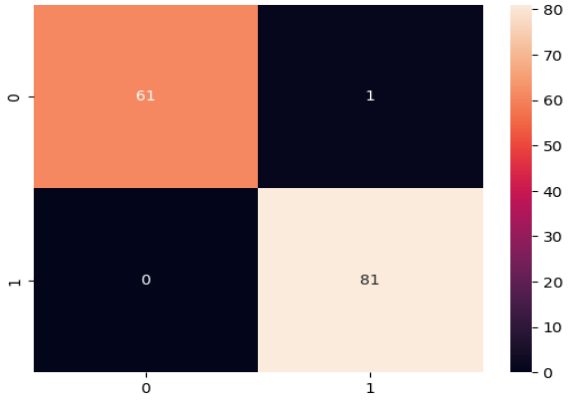


Fig 17. confusion matrix

Metrics evaluation: fig.18 Demonstrate the evaluation of suggested BiLSTM-CNN model in terms of accuracy, precision, recall and f1-score.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        62
           1       0.99      1.00      0.99        81

    accuracy                           0.99       143
   macro avg       0.99      0.99      0.99       143
weighted avg       0.99      0.99      0.99       143
```

Fig 18. Metrics evaluation.

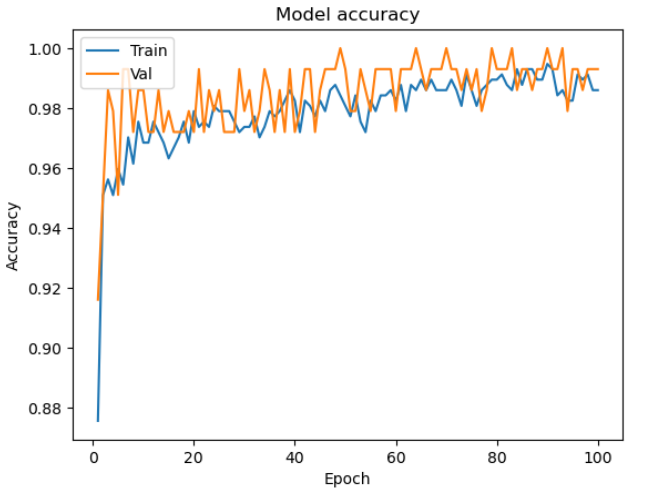
Accuracy versus epochs and Loss versus epochs plots: The plot accuracy versus epochs shows the capability of the model to correctly classify over a training set and provide information about

Model Convergence and the ability for overfitting or underfitting. Also, loss versus epochs plot gives insight into model optimization and model capability to reduce mistakes in prediction.

```
def plot_learningCurve(history, epoch):
  # Plot training & validation accuracy values
  epoch_range = range(1, epoch+1)
  plt.plot(epoch_range, history.history['accuracy'])
  plt.plot(epoch_range, history.history['val_accuracy'])
  plt.title('Model accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()

  # Plot training & validation loss values
  plt.plot(epoch_range, history.history['loss'])
  plt.plot(epoch_range, history.history['val_loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()
```

```
epochs=100
plot_learningCurve(history, epochs)
```



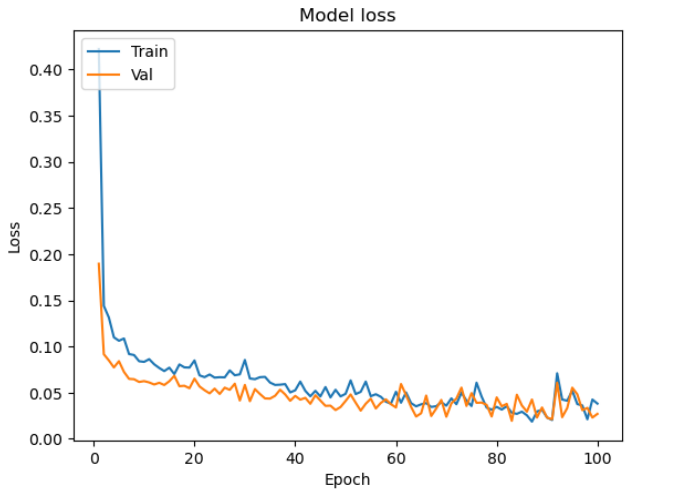


Fig 19. Accuracy versus epochs and Loss versus epochs plots

Applying Explainable AI using Shapley Additive Explanations (SHAP)

```
import shap


# Reshape X_train and X_test
X_train_reshape = X_train.reshape(571, 18)  # Reshape to 2 dimensions
X_test_reshape = X_test.reshape(143, 18)  # Reshape to 2 dimensions

# Create a SHAP explainer object
explainer = shap.Explainer(model, X_train_reshape)

# Calculate SHAP values for a subset of the data (e.g., X_test_reshape)
shap_values = explainer.shap_values(X_test_reshape)


#X_train = X_train.reshape(571,18,1)
#X_test = X_test.reshape(143, 18, 1)
```

```
PermutationExplainer explainer: 144it [08:26,  3.54s/it]
```

Summary Plot using SHAP Explanation

```
# Plot the SHAP values
shap.summary_plot(shap_values, X_test1)
```

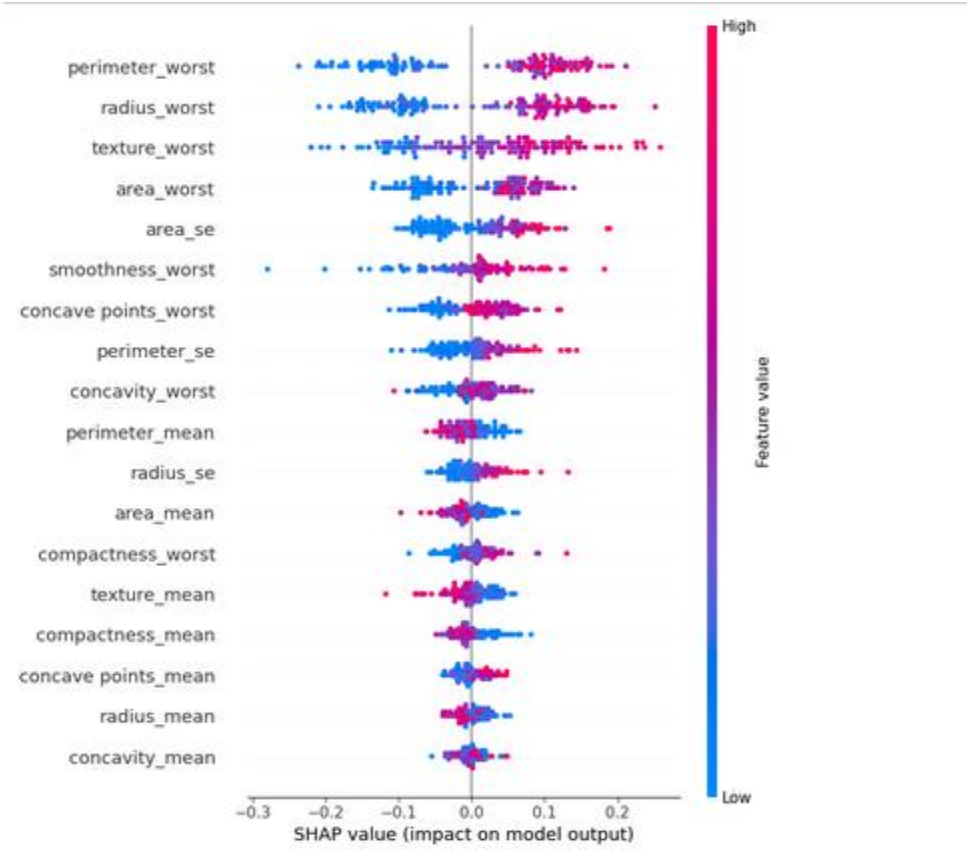


Fig 20. Summary Plot using SHAP Explanation.

Generated Local Explanation of signal instance using waterfall plot

```
# Select a specific instance from the dataset (e.g., the first instance)
instance_index = 1  # Change this index to visualize a different instance
shap_values_instance = shap_values[instance_index]

# Calculate the baseline value (e.g., using the mean of model predictions)
baseline_value = np.mean(model.predict(X_test))

# Create an Explanation object for the selected instance
shap_values_explanation = shap.Explanation(values=shap_values_instance, base_values=baseline_value, data=X_test1.iloc[instance_index])

# Create a waterfall plot using the SHAP values of the selected instance
shap.plots.waterfall(shap_values_explanation)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

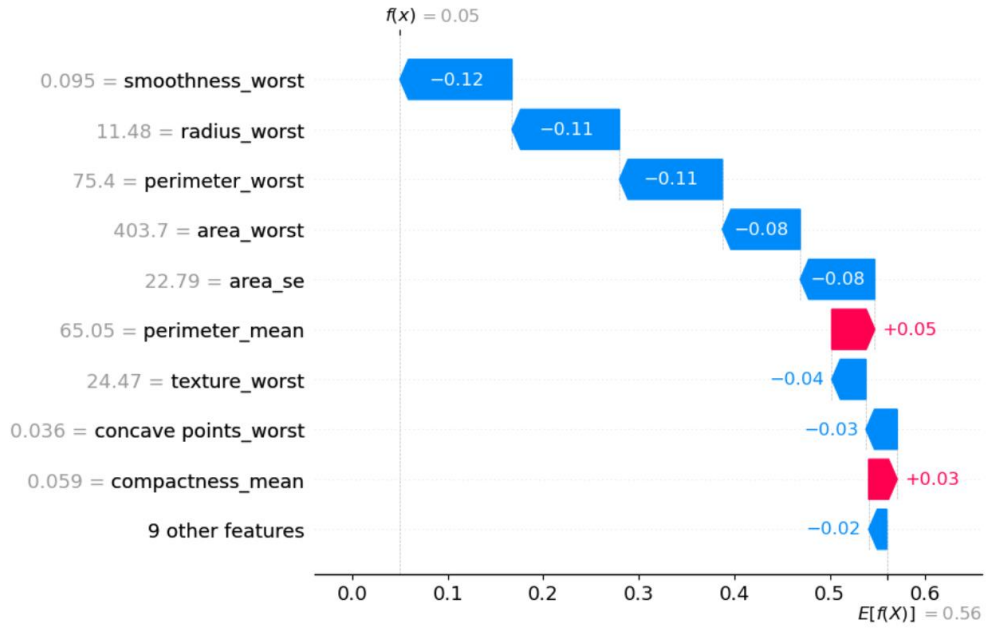


Fig 21. Generated Local Explanation of signal instance using waterfall plot

Generated Local Explanation of signal instance using Force plot

```
instance_index = 100  # Change this index to visualize a different instance
shap_values_instance = shap_values[instance_index]
baseline_value = np.mean(model.predict(X_test))

# Create an Explanation object for the selected instance
shap_values_explanation = shap.Explanation(values=shap_values_instance, base_values=baseline_value, data=X_test1.iloc[instance_index])

shap.initjs()
shap.force_plot(shap_values_explanation,
shap_values[instance_index],
X_test1.iloc[instance_index])
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
```

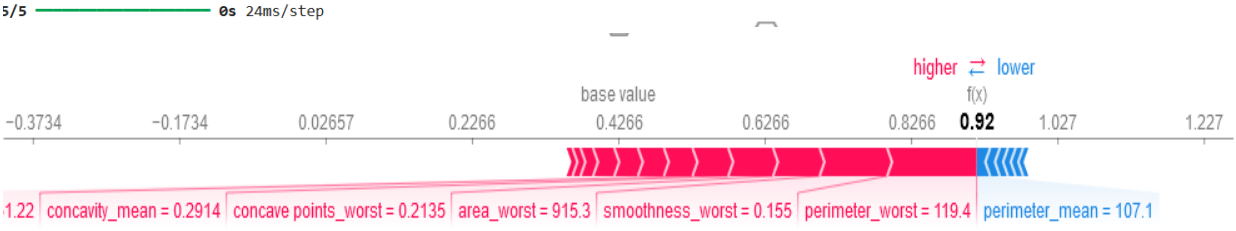


Fig 22. Generated Local Explanation of signal instance using Force plot

SHAP plot generated Global Explanation.

```
# Plot the SHAP values
shap.force_plot(baseline_value, shap_values, X_test1)
```

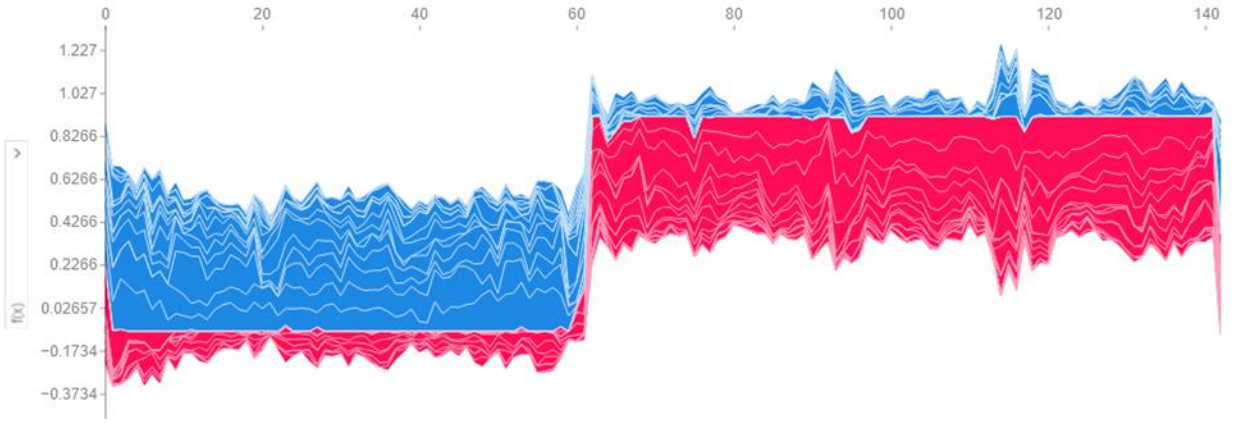


Fig 23. SHAP plot generated Global Explanation