The experimental results showcase the accuracy of both the B12 FRCNN and Kai-BiLSTM m–odels. Additionally, they highlight the comparative performance of the existing system, which combines skeletal image feature extraction via Convolutional Neural Network (CNN) and textual or question answer feature extraction using Long Short Term Memory (LSTM) models, exhibiting an accuracy of 83.9 percent across visual and textual datasets.

Existing feature detection algorithms, leveraging deep belief networks (DBN) and LSTM, demonstrate an accuracy rate of 85.9%. Various models employing LSTM for textual feature analysis and Recurrent Neural Networks (RNN) for feature extraction achieve an accuracy of 89.1946. Furthermore, the accuracy of other existing models, such as CNN and BiLSTM, analyzing visual and textual data, reaches 91.222%.

There is potential for further precision enhancement in the proposed model by leveraging updated datasets. Leveraging the new B12 FRCNN (Block12 Faster RCNN) model and the Kai-BiLSTM model, introduced with a remarkable 96.9% accuracy in this study, could significantly contribute to this improvement.

4.1 Experimental Result on VQA

The CLEF initiative labs are organizing the CLEF Image Retrieval and Classification Task 2019 campaign, inviting teams worldwide to participate in various research tasks. To ensure a focused evaluation, the questions are categorized based on modality, plane, organ system, and abnormality. These categories aim to challenge text creation and classification techniques effectively. Specifically, medical questions in this VQA challenge focus on individual characteristics, allowing for assessment solely based on visual content, without requiring specialized medical expertise.

The Healthcare Visual Q&A 2019 Training Set provides the most frequent answers for each category. For instance, modality options include xr-plain film, t2, us-ultrasound, and more. Similarly, the plane category lists axial, sagittal, coronal, and other planes. The organ system category comprises responses like skull and contents, musculoskeletal, gastrointestinal, and others. Finally, the abnormality category includes responses such as yes, no, meningioma, glioblastoma multiforme, and more.

To uphold precision, the responses generated during testing underwent manual validation by both a physician and a radiologist. A total of thirty-three responses were adjusted, primarily to incorporate optional elements, enhance the range of viable responses, or refine automated replies. Because the training and validation sets were created using the same data generation procedures, the error rate should be similar. The test set includes 500 medical images and 500 related questions.

Evaluation metrics are crucial for assessing the performance, efficiency, and success of a system, process, or strategy. These metrics, which can be statistical or interpretive, serve as accurate measurements or indicators and are often based on key performance indicators (KPIs). They are widely utilized across various domains such as business, marketing, healthcare, education, and technology to evaluate the effectiveness of strategies or procedures, identify areas for improvement, and derive insights from collected data. In the realm of VQA research, a key goal is to develop computer vision systems capable of performing diverse tasks rather than specializing in just one area like object recognition.

The Precision metric in Visual Question Answering (VQA) quantifies the proportion of questions within a dataset for which the model generates correct answers. In VQA, the system receives an image along with a natural language question and is tasked with producing a suitable response.

In the context of Visual QA (VQA), the accuracy metric is often calculated by dividing the total number of right answers by the total number of questions in the dataset. For example, if a VQA model correctly answers 800 out of 1,000 questions in a dataset, its accuracy will be 80%. Accuracy is a crucial evaluation metric that indicates how well a model understands visual content and responds to related questions. To acquire a more complete knowledge of a model's performance, additional measures such as precision, recall, and F1 score are conceivably used. These measures provide nuanced insights beyond accuracy, allowing for a more complete assessment of the model's capabilities. The assessed accuracy of our model remains at approximately 50%.

By comparing generated translations to a reference translation, the metric known as BLEU (Bilingual Evaluation Understudy) is frequently used in machine translation to assess the standard of the output. To assess the effectiveness of visual question-answering (VQA) systems, BLEU has also been modified.

The BLEU score in VQA evaluates how closely the system's generated responses correspond to the reference responses given in the dataset. For each question-answer combination, the metric is first calculated by altering the n-gram precision, which quantifies the overlap of n-gram sequences between the generated and reference answers. The geometric mean of the n-gram precisions for each question in the dataset is then computed using the revised n-gram precision.

A higher BLEU (0-1) score indicates superior performance. Although the BLEU score can offer some indication of the quality of outputs from a VQA system, it is crucial to acknowledge its substantial limitations. These limitations include its failure to capture semantic similarity across responses and its inability to consider the diversity of valid answers to a given question. Consequently, it is advisable to employ a range of evaluation measures, including BLEU, to obtain a comprehensive understanding of the effectiveness of a VQA system.

$$min(1-\frac{r}{c},0)+\sum_{n=1}^{4} \frac{log\ p_n}{4} \quad (12)$$

where:

The reference_length attribute denotes the length of the reference answer, while the output_length attribute specifies the length of the generated answer. Blue_ngram_weights refer to the weights utilized for computing n-gram precisions, where pn signifies the n-gram precision for n-grams of length n in the generated response. Here, n represents the length of the n-grams employed to determine precision.

The weights utilized to compute n-gram precisions are typically predetermined, although they can alternatively be derived from data. For instance, if unigrams (n=1) and bigrams (n=2) are selected, the weights may be assigned as [0.5, 0.5] to evenly distribute the significance of each precision.

The geometric mean of the BLEU score is calculated by aggregating the corrected n-gram precisions obtained from each item in the dataset. The discrepancy in length between the reference and generated responses is factored in during the computation of the adjusted n-gram precision. Specifically, it is computed as the exponentiation of the arithmetic mean of the log-transformed n-gram precisions. The outcome analysis of the first phase dataset Healthcare Visual Q&A 2019 for each type of Visual Question Answering System.

	Actually Positive (1)	**Actually Negative (0)**
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Fig 34 : Confusion Matrix

The F-measure, often known as the F-score or F1 metric when the β value is 1, is a weighted harmonic mean of Recall and Precision. This metric is utilized for several reasons. The harmonic mean is typically the appropriate choice when averaging rates or frequencies. Additionally, a set-theoretic rationale for its use will be addressed subsequently. The more general form denoted as F allows for variable weighting of Recall and Precision, although it is common practice to assign them equal weight, resulting in the F1 score, which is the prevalent reference when discussing the F-measure.

A variant of accuracy not affected by negatives, single value measures(compare, tune systems). Harmonic mean of P and R is mentioned in equation (12)

$$F_\beta = \frac{(\beta^2+1).P.R}{\beta^2 P+R} \tag{13}$$

where $\beta = 1$, which gives

$$F_1 = \frac{2PR}{P+R} \tag{14}$$

Geometric interpretation the percentage overlap between relevant and retrieved which followed by

$$F_1 = \frac{2PR}{P+R} = 2\left(\frac{1}{P} + \frac{1}{R}\right)^{-1} \tag{15}$$

$$F_1 = 2\left(\frac{TP+FP}{TP} + \frac{TP+FN}{TP}\right)^{-1} = 2\,\frac{rel.ret}{rel+ret} \tag{16}$$

Precision is a statistical metric utilized to assess the accuracy of positive predictions generated by a classifier. It is calculated as the quotient of true positive predictions divided by the total number of positive predictions made by the classifier, irrespective of their correctness.

The formula for precision is:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Another way to describe accuracy is as the percentage of correctly predicted positive cases (true positives) out of all instances identified as positive by the classifier.

Here's a breakdown of the terms used in the formula:

- True Positives (TP): The number of occurrences accurately recognised as positive by the classifier that are also true positives.
- False Positives (FP): The number of cases that the classifier wrongly classified as positive despite being negative.

To calculate precision, you need to count the number of true positives and false positives from the classifier's predictions and then plug them into the formula. The calculated ratio will fall within the range of 0 to 1, where higher values correspond to higher precision, indicating fewer false positive predictions made by the classifier.

Recall, also known as sensitivity or true positive rate, is a statistic that assesses a classifier's ability to properly identify positive occurrences among all actual positive examples in a dataset. It calculates the proportion of true positive predictions made by the classifier compared to the total number of actual positive cases.

The formula for recall is:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Recall is defined as the ratio of accurately predicted positive cases (true positives) to total positive instances (true positives + false negatives).

Here's a breakdown of the terms used in the formula:

- True Positives (TP): This refers to the count of instances that are genuinely positive and are accurately identified as such by the classifier.
- False Negatives (FN): This indicates the count of positive occurrences that the classifier incorrectly identifies as negative.

To calculate recall, you need to count the number of true positives and false negatives from the classifier's predictions and then plug them into the formula. The resulting value will be a ratio between 0 and 1, where a higher value indicates better recall (i.e., fewer false negatives).

The F1 score, often known as the F-measure or F-score, is a metric for assessing the effectiveness of a classification model. It takes into account both the model's precision and recall in order to compute a single score that balances the trade-offs.

The F1 score is calculated using the following formula:

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

Where:

- Precision is the percentage of accurate positive predictions from all positive predictions provided by the model. It assesses the accuracy of optimistic predictions.
- Recall is the proportion of true positive predictions among all positive instances in the dataset. It evaluates the model's ability to correctly identify positive cases.
- The F1 score represents the harmonic mean of precision and recall. It ranges from 0 to 1, with higher values indicating better performance.

In classification, sensitivity, also referred to as recall or true positive rate, evaluates the classifier's capability to correctly identify positive instances from the entirety of actual positive examples within the dataset. Sensitivity holds particular importance in scenarios where the cost associated with overlooking positive examples (false negatives) is significant. It serves as a metric to gauge the classifier's effectiveness in capturing all pertinent instances of a specified class.

Mathematically, sensitivity is calculated using the following formula:

$$Sensitivity\ (Recall) = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

- True Positives (TP) are instances that the model effectively classifies as positive.
- False Negatives (FN) are situations that are genuinely positive but are misclassified as negative by the model.

Sensitivity is a fraction that runs between 0 and 1, with a number closer to 1 indicating more sensitivity or memory. A sensitivity of one suggests that the classifier properly recognises all positive examples, whereas a sensitivity of zero indicates that the classifier fails to identify any positive instances.

Sensitivity is widely utilized in medical diagnostics, anomaly detection, and other applications where identifying true positives is critical.

Classification specificity evaluates a classifier's capacity to accurately recognize negative instances among all genuine negative examples in a dataset. It serves as a complement to the false positive rate and is particularly advantageous in scenarios where the consequences of false alarms (false positives) are significant.

Mathematically, specificity is calculated using the following formula:

$$Classification\ Specificity = \frac{True\ Negative}{True\ Negative + False\ Positive}$$

Where:

- True Negatives (TN) are events that were accurately categorized as negative by the model.
- False Positives (FP) are events that are truly negative but are misclassified as positive by the model.

Specificity, represented as a fraction between 0 and 1, signifies the degree of accuracy in identifying negative instances by the classifier. A value closer to 1 suggests higher specificity, indicating that the classifier adeptly detects all negative occurrences. Conversely, a specificity value nearing zero implies the classifier fails to identify any negative examples. This metric holds significance in various domains, including medical diagnostics and spam detection, where minimizing false alarms is paramount for effective decision-making.

Table 6 : Most Frequent Answers for various Question Type and Answer count

Question Types	Most Frequent Answers	Total No. of Answers
MODALITY	No	554
	Yes	552
	xr-plain film	456
	t2	217
	us-ultrasound	183
	t1	137
	constrast	107
	noncontrast	102
	ct non contrast	84
PLANES	axil	1558
	sagittal	478
	coronal	389
	ap	197
	lateral	151
	frontal	120
	pa	92
	transverse	76
	oblique	50
ORGAN SYSTEM	genitourinary	214
	face, sinuses and neck	191
	vascular and lymphatic	122
	heart and great vessels	120
	breast	65
	muscloskeletal	438
	Yes	62
	No	48

Question Types	Most Frequent Answers	Total No. of Answers
ABNORMALITY	meningioma	30
	glioblastoma multiforme	28
	pulmonary embolism	16
	acute appendicitis	14
	arteriovenous malformation (avm)	14
	arachnoid cyst	13
	schwannoma	13
	tuberous sclerosis	12
	brain, cerebral abscess	12
	ependymoma	12
	fibrous dysplasia	12
	multiple sclerosis	12
	diverticulitis	11
	langerhan cell histiocytosis	11
	sarcoidosis	11

Fig 33 : Most frequent answers from different question types

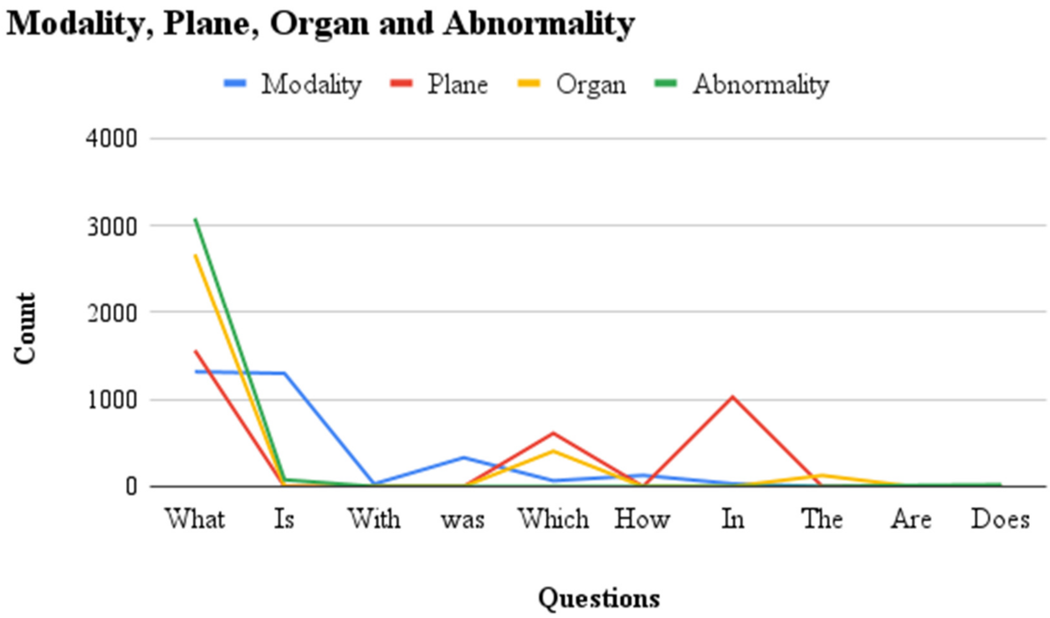

Fig 36 : Different methods of question for various question types

Table 7 : Total count of Visual and Textual dataset

Images	**3200**
No. of Questions and Answer	12792

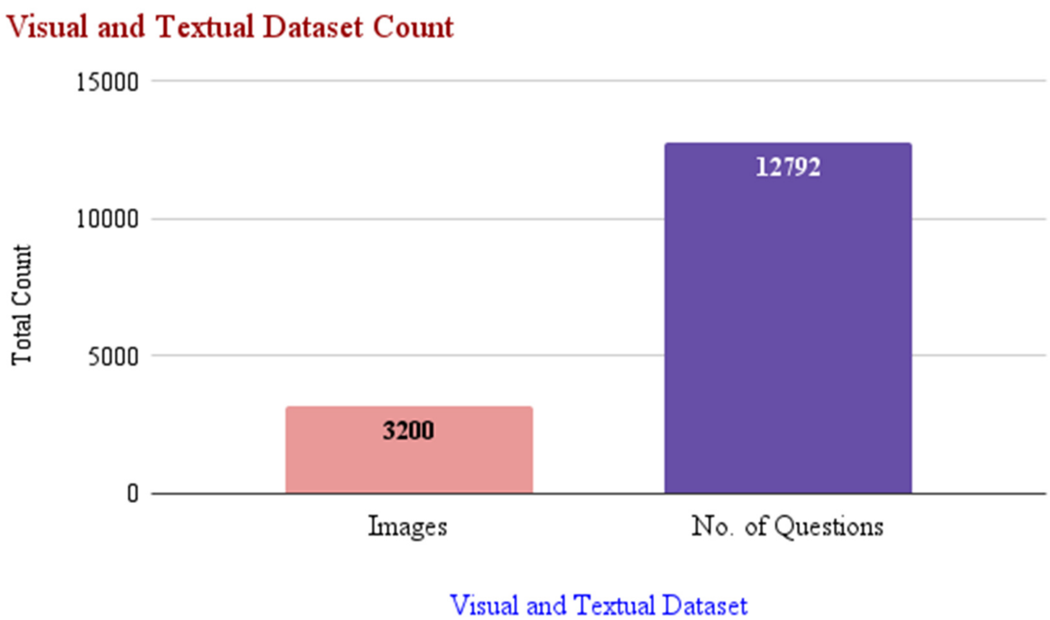

Fig 37 : Total count of Image and questions from CLEF Image Retrieval and Classification Task 2019 Dataset

The table provided indicates the distribution of questions and answers across different types of questions, namely Modality, Plane, Organ, and Abnormality. Here's a detailed explanation:

1. **Modality**: This category pertains to questions related to the imaging modality used to capture medical images, such as X-ray, MRI, CT scan, ultrasound, etc. The table indicates that there are a total of 3200 questions and answers associated with the Modality category.
2. **Plane**: Refers to questions concerning the imaging plane or orientation of the captured medical images, such as axial, sagittal, coronal, etc. Similar to the Modality category, there are also 3200 questions and answers related to the Plane category.
3. **Organ**: Represents questions regarding specific organs or organ systems depicted in the medical images. Examples include questions about the brain, lungs, heart, musculoskeletal system, etc. Again, there are 3200 questions and answers allocated to the Organ category.
4. **Abnormality**: Denotes questions pertaining to the identification or diagnosis of abnormalities or pathologies present in the medical images. This could include conditions like tumors, fractures, infections, etc. There are slightly fewer questions and answers in this category, totaling 3192.

Overall, each category has an equal number of questions and answers, except for the Abnormality category, which has a slightly lower count. These questions and answers are crucial for training and evaluating models in medical image analysis and interpretation tasks, contributing to advancements in healthcare and diagnostic capabilities.

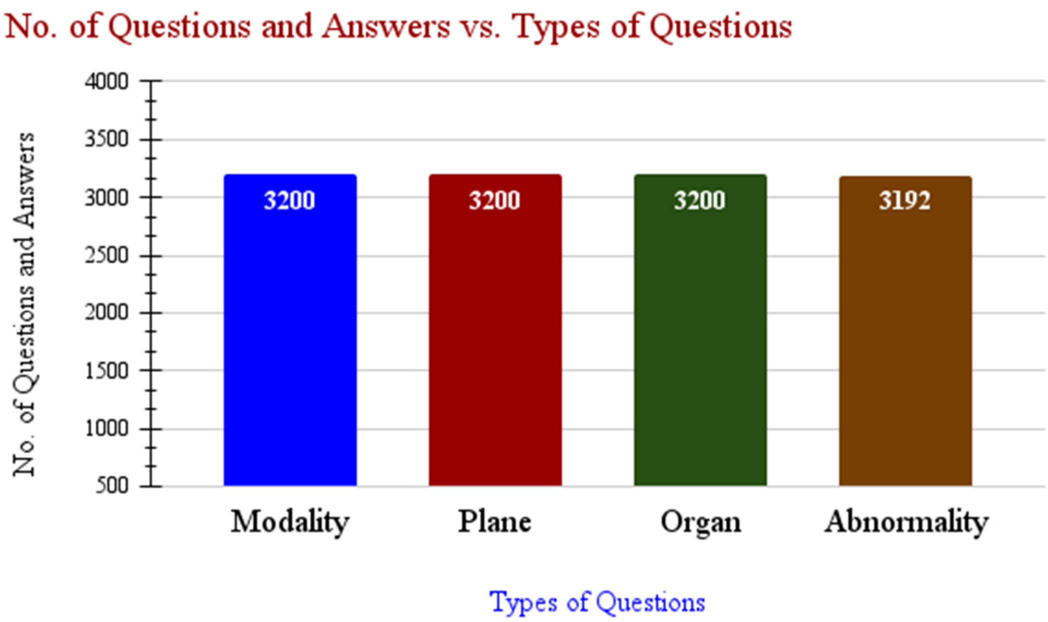

Fig 38 : Total number of data from both question and answer for various types

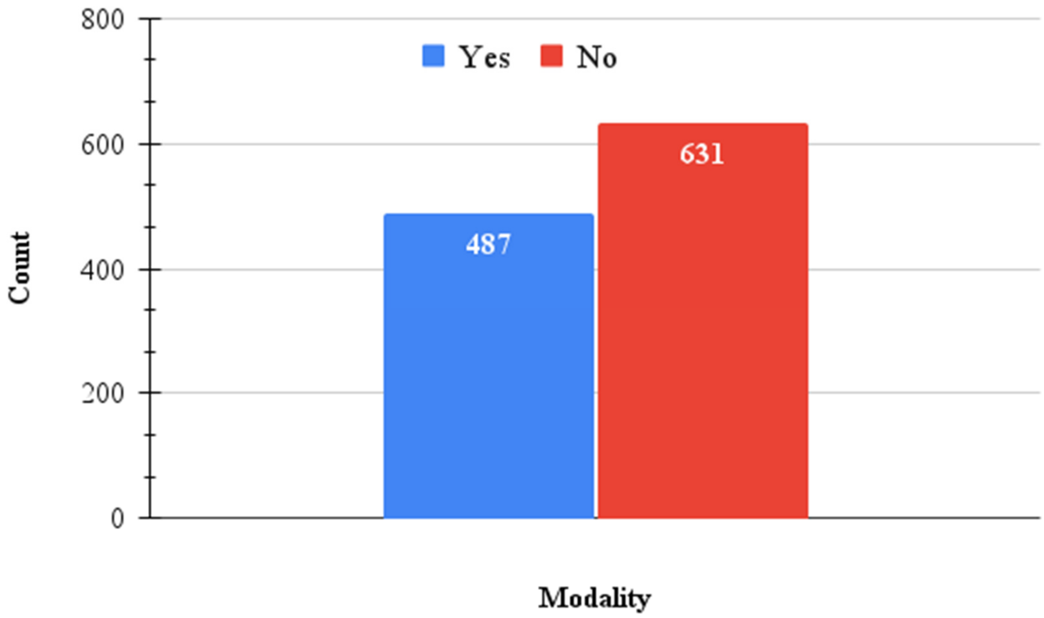

Fig 39 : Count of boolean questions for Modality

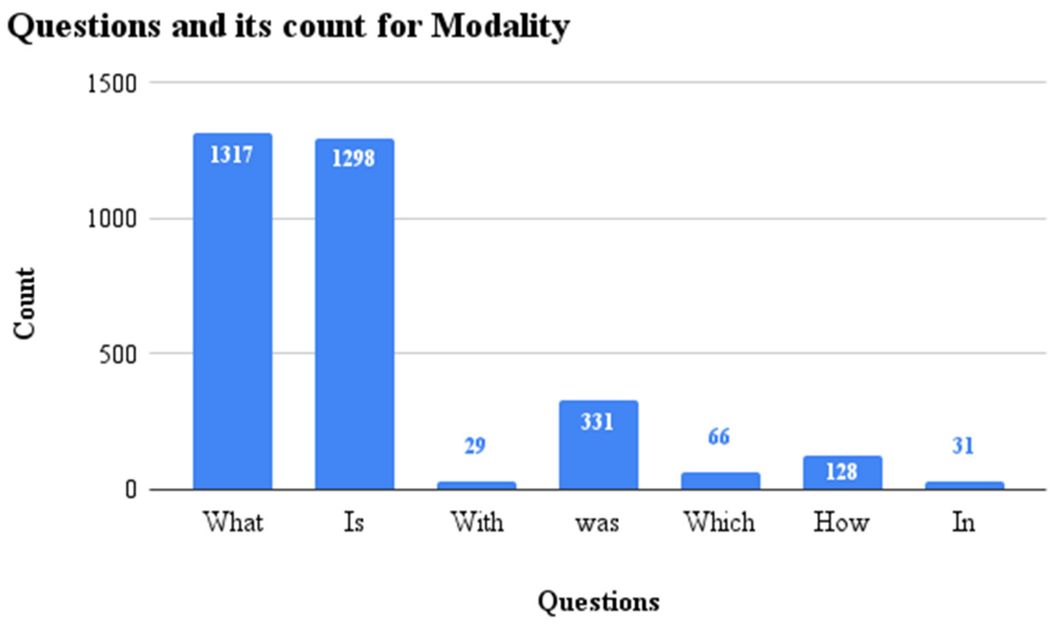

Fig 40 : Various types of question for Modality question answering data type

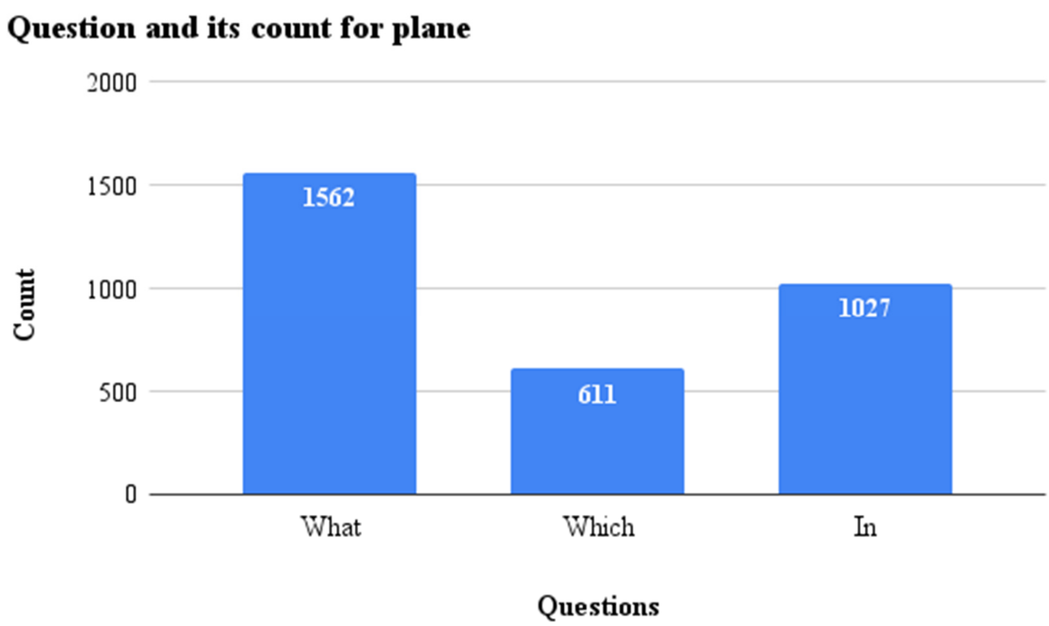

Fig 41 : Various types of question for Plane question answering data type

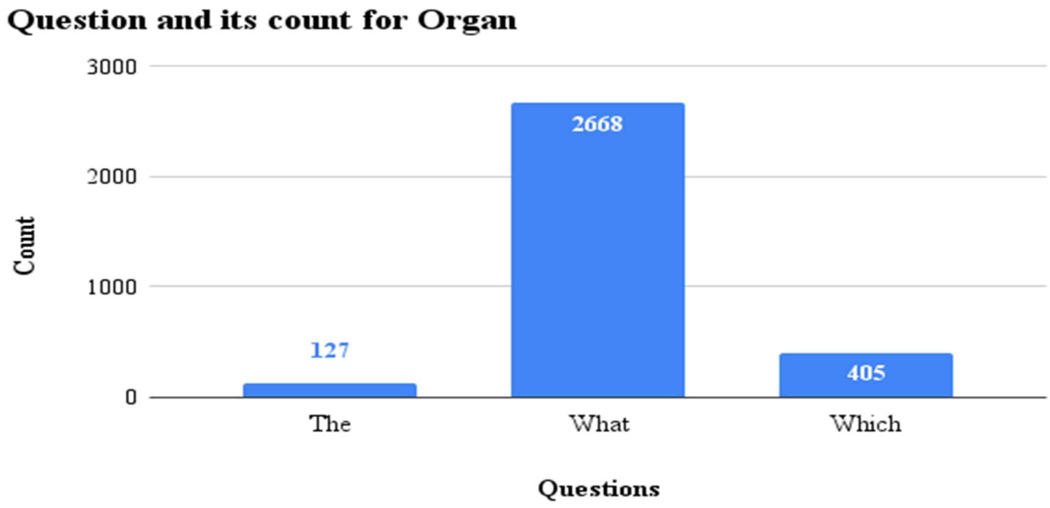

Fig 42 : Various types of question for Organ question answering data type

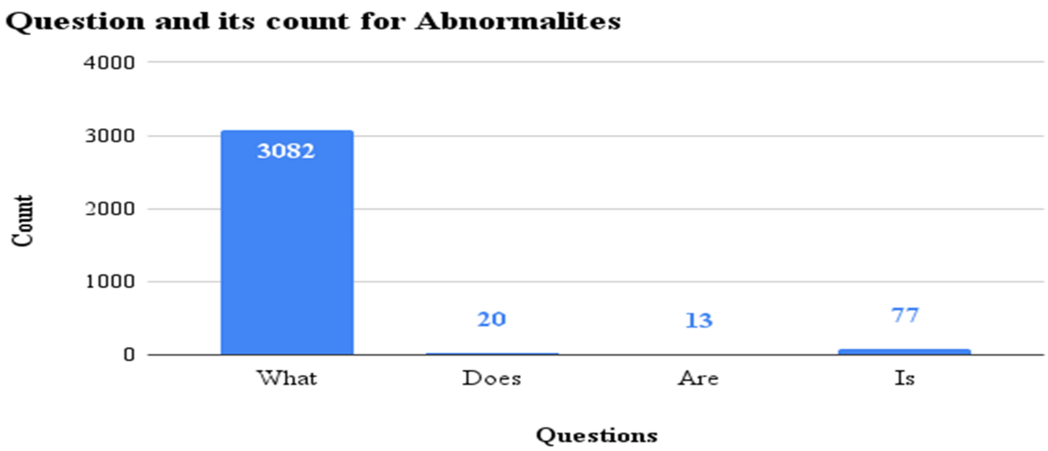

Fig 43 : Various types of question for Abnormality question answering data type

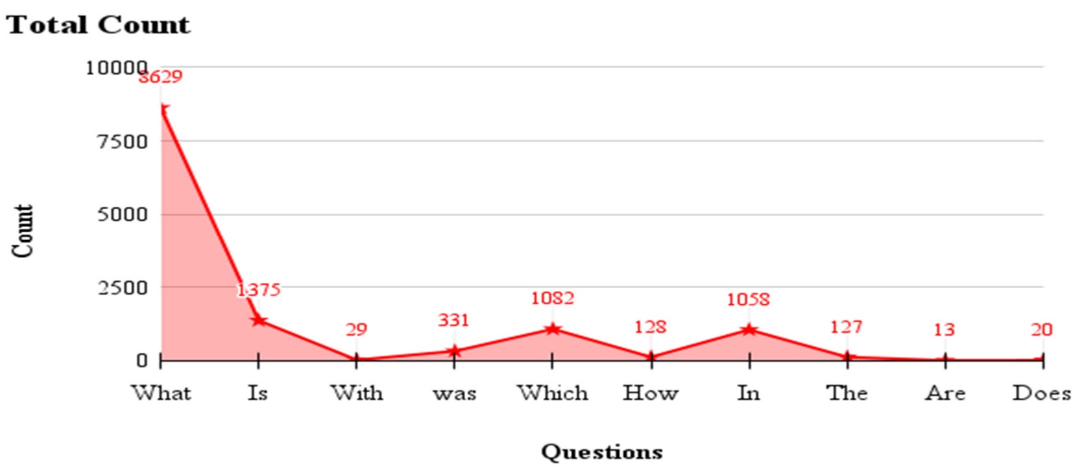

Fig 44 : Overall count of different question types

```
supported_activation_functions = ("sigmoid", "relu", "softmax")

def sigmoid(self, sop):
    if type(sop) in [list, tuple]:
        sop = numpy.array(sop)

    return 1.0 / (1 + numpy.exp(-1 * sop))

def relu(self, sop):
    if not (type(sop) in [list, tuple, numpy.ndarray]):
        if sop < 0:
            return 0
        else:
            return sop
    elif type(sop) in [list, tuple]:
        sop = numpy.array(sop)

    result = sop
    result[sop < 0] = 0

    return result

def softmax(self, layer_outputs):
    return layer_outputs / (numpy.sum(layer_outputs) + 0.000001)
```

```
def layers_weights(self, model, initial=True):
    network_weights = []

    layer = model.last_layer
    while "previous_layer" in layer.__init__.__code__.co_varnames:
        if type(layer) in [self.Conv2D, self.Dense]:
            if initial == True:
                network_weights.append(layer.initial_weights)
            elif initial == False:
                network_weights.append(layer.trained_weights)
            else:
                raise ValueError("Unexpected value to the 'initial' parameter:
nitial}.".format(initial=initial))

        # Go to the previous layer.
        layer = layer.previous_layer

    # If the first layer in the network is not an input layer (i.e. an instance of the Input2D class),
aise an error.
    if not (type(layer) is self.Input2D):
        raise TypeError("The first layer in the network architecture must be an input layer.")

    network_weights.reverse()
    return numpy.array(network_weights)
```

```
            weights_vector = vector_weights[start:start + layer_weights_size]
            #     matrix = pygad.nn.DenseLayer.to_array(vector=weights_vector,
shape=layer_weights_shape)
            matrix = numpy.reshape(weights_vector, newshape=(layer_weights_shape))
            network_weights.append(matrix)

            start = start + layer_weights_size

        # Go to the previous layer.
        layer = layer.previous_layer

    # If the first layer in the network is not an input layer (i.e. an instance of the Input2D class),
raise an error.
    if not (type(layer) is self.Input2D):
        raise TypeError("The first layer in the network architecture must be an input layer.")

    network_weights.reverse()
    return numpy.array(network_weights)

def layers_weights_as_vector(self, model, initial=True):
    network_weights = []

    layer = model.last_layer
    while "previous_layer" in layer.__init__.__code__.co_varnames:
        if type(layer) in [self.Conv2D, self.Dense]:
            # If the 'initial' parameter is True, append the initial weights. Otherwise, append the trained
weights.
```

```
    if not (type(layer) is self.Input2D):
        raise TypeError("The first layer in the network architecture must be an input layer

    network_weights.reverse()
    return numpy.array(network_weights)

def update_layers_trained_weights(self, model, final_weights):
    layer = model.last_layer
    layer_idx = len(final_weights) - 1
    while "previous_layer" in layer.__init__.__code__.co_varnames:
        if type(layer) in [self.Conv2D, self.Dense]:
            layer.trained_weights = final_weights[layer_idx]

            layer_idx = layer_idx - 1

        # Go to the previous layer.
        layer = layer.previous_layer
```

Fig 45 : Sample code function on Existing CNN algorithm

4.2 Code Implementation

This Python class ExistingCNN contains several methods related to a convolutional neural network (CNN) model, such as activation functions (sigmoid, relu, softmax), weight manipulation, and updating trained weights. Here's a brief explanation of each function:

1. Activation Functions: The class provides implementations for common activation functions used in neural networks, including sigmoid, ReLU, and softmax.
2. layers_weights: This method extracts the weights of all layers in the model and returns them as an array. It can return either the initial weights or the trained weights depending on the initial parameter.

3. layers_weights_as_matrix: Similar to layers_weights, but it returns the weights of each layer reshaped as a matrix instead of an array.
4. layers_weights_as_vector: Similar to layers_weights, but it returns the weights of each layer flattened into a vector.
5. update_layers_trained_weights: This method updates the trained weights of each layer in the model using the provided final_weights array.

```
    # construct DBN
    dbn = ExistingDBN(input=x, label=y, n_ins=6, hidden_layer_sizes=[3, 3], n_outs=2, rng=rng)

    # pre-training (TrainUnsupervisedDBN)
    dbn.pretrain(lr=pretrain_lr, k=1, epochs=pretraining_epochs)

    # fine-tuning (DBNSupervisedFineTuning)
    dbn.finetune(lr=finetune_lr, epochs=finetune_epochs)

    # test
    x = numpy.array([[1, 1, 0, 0, 0, 0],
                     [0, 0, 0, 1, 1, 0],
                     [1, 1, 1, 1, 1, 0]])

    # print
    dbn.predict(x)

def training(self, iptrdata):
    parser = argparse.ArgumentParser(description='Train')

    parser.add_argument('-train', help='Train data', type=str, required=True)
    parser.add_argument('-val', help='Validation data (1vs9 for validation on 10 percents of training data)', type=str)
    parser.add_argument('-test', help='Test data', type=str)

    parser.add_argument('-e', help='Number of epochs', type=int, default=1000)
    parser.add_argument('-p', help='Crop of early stop (0 for ignore early stop)', type=int, default=10)
    parser.add_argument('-b', help='Batch size', type=int, default=128)

    parser.add_argument('-pre', help='Pre-trained weight', type=str)
    parser.add_argument('-name', help='Saved model name', type=str, required=True)
```

```
train_inputs = []
train_outputs = []
time.sleep(78)
if len(train_inputs) > 0:
    if (train_inputs.ndim != 4):
        raise ValueError("The training data input has {num_dims} but it must have 4 dimensions. The first dimension is the number of t
    if (train_inputs.shape[0] != len(train_outputs)):
        raise ValueError(
                "Mismatch between the number of input samples and number of labels: {num_samples_inputs} != {num_samples_outputs}.

    network_predictions = []
    network_error = 0
    for epoch in range(self.epochs):
        print("Epoch {epoch}".format(epoch=epoch))
        for sample_idx in range(train_inputs.shape[0]):
            # print("Sample {sample_idx}".format(sample_idx=sample_idx))
            self.feed_sample(train_inputs[sample_idx, :])

            try:
                predicted_label = \
                    self.numpy.where(self.numpy.max(self.last_layer.layer_output) == self.last_layer.layer_output)[0][0]
            except IndexError:
                print(self.last_layer.layer_output)
                raise IndexError("Index out of range")
            network_predictions.append(predicted_label)

            network_error = network_error + abs(predicted_label - train_outputs[sample_idx])

            self.update_weights(network_error)
```

Fig 46 : Construction of Existing DBN structure for Medical Images

1. **DBN Initialization**: An illustration of the ExistingDBN class is delivered with specified parameters such as input size (n_ins), number of hidden layers and their sizes (hidden_layer_sizes), output size (n_outs), and random number generator (rng).
2. **Pre-training**: The DBN is pre-trained using unsupervised learning (contrastive divergence algorithm) to learn the weights in an unsupervised manner. The learning rate (pretrain_lr) and number of pre-training epochs (pretraining_epochs) are specified.
3. **Fine-tuning**: After pre-training, the DBN is fine-tuned using supervised learning (backpropagation) to adjust the weights based on labeled data. The learning rate (finetune_lr) and number of fine-tuning epochs (finetune_epochs) are specified.
4. **Testing**: Test data (x) is provided to the trained DBN, and predictions are made using the predict method of the ExistingDBN class.
5. **Training Method**: The training method is defined, which appears to be used for training the DBN model. It parses command-line arguments for training data, validation data, test data, number of epochs, batch size, pre-trained weights, and saved model name.
6. **Data Validation**: The code checks if the training inputs have the correct dimensions and if the number of input samples matches the number of labels. If not, it raises a ValueError.
7. **Training Loop**: The code iterates over epochs and samples, feeds each sample to the network, makes predictions, calculates network error, and updates the weights based on the error using the update_weights method.

```
class Model:
        def __init__(self, last_layer, epochs=10, learning_rate=0.01):

            self.last_layer = last_layer
            self.epochs = epochs
            self.learning_rate = learning_rate

            # The network_layers attribute is a list holding references to all CNN layers.
            self.network_layers = self.get_layers()

        def get_layers(self):

            network_layers = []

            layer = self.last_layer

            while "previous_layer" in layer.__init__.__code__.co_varnames:
                network_layers.insert(0, layer)
                layer = layer.previous_layer

            return network_layers

        def train(self, train_inputs, train_outputs):

            if (train_inputs.ndim != 4):
                raise ValueError(
                    "The training data input has {num_dims} but it must have 4 dimensions. The first dimension is the number of t
                        num_dims=train_inputs.ndim))

            if (train_inputs.shape[0] != len(train_outputs)):
                raise ValueError(
                    "Mismatch between the number of input samples and number of labels: {num_samples_inputs} != {num_samples_outp
                        num_samples_inputs=train_inputs.shape[0], num_samples_outputs=len(train_outputs)))

            network_predictions = []
            network_error = 0

            for epoch in range(self.epochs):
                print("Epoch {epoch}".format(epoch=epoch))
                for sample_idx in range(train_inputs.shape[0]):
                    # print("Sample {sample_idx}".format(sample_idx=sample_idx))
                    self.feed_sample(train_inputs[sample_idx, :])

                    try:
                        predicted_label = \
                            numpy.where(numpy.max(self.last_layer.layer_output) == self.last_layer.layer_output)[0][0]
                    except IndexError:
                        print(self.last_layer.layer_output)
                        raise IndexError("Index out of range")
                    network_predictions.append(predicted_label)
```

Fig 47 : Code function on Proposed Kai_BiLSTM

4.3 Comparison analysis of Feature extraction techniques

Table 8 : The table presents the performance metrics for a certain method across different evaluation criteria

Method	Accuracy
Manhattan Distance (MD)	50.40%
Euclidean Distance (ED)	57.80%
Jaccard Similarity Coefficient (JSC)	58.90%
Cosine Similarity (CS)	60.10%

This metric computes the ratio of correct predictions generated by the method. It quantifies the spatial separation between two points within a grid-based framework by summing the absolute disparities in their respective coordinates. Here, the method achieved an accuracy of 50.40% when evaluated using Manhattan Distance. Euclidean Distance metric calculates the straight-line distance between two points in space. The method achieved an accuracy of 57.80% Jaccard Similarity Coefficient measures the similarity between two sets by comparing their intersection to their union. The method achieved an accuracy of 58.90% Cosine Similarity: This metric measures the angle between two vectors, indicating their similarity. The method achieved an accuracy of 60.10% as denoted in Table 8.

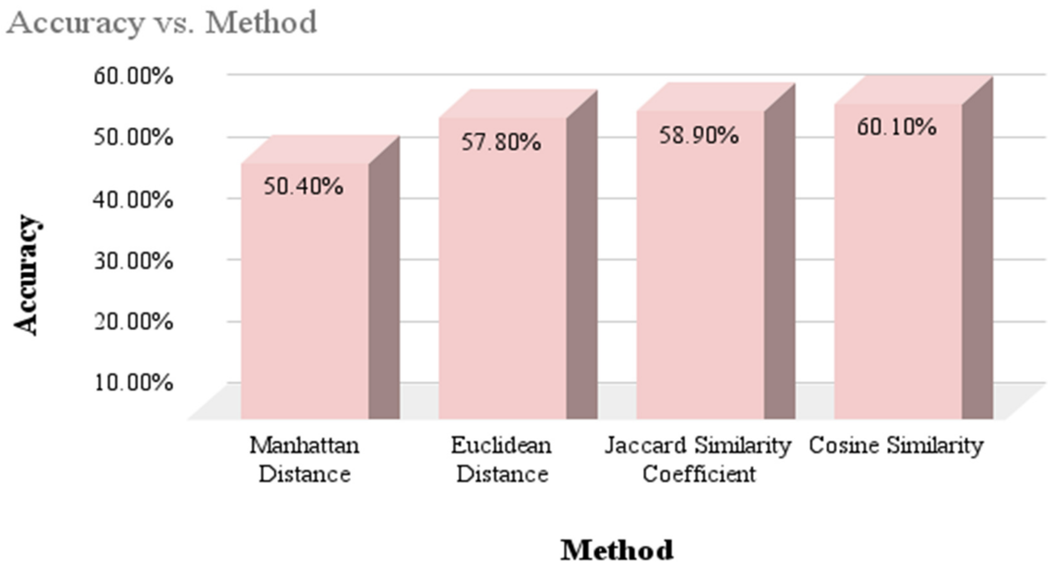

Fig 48 : The performance metrics for a certain method across different evaluation criteria

The KNN Classifier demonstrated an accuracy of 28.20 percent. KNN, or K-Nearest Neighbours, presents a straightforward approach to classifying data points by assigning them to the majority class among their nearest neighbors.

The Soft-Max Classifier achieved an accuracy of 34.10 percent. Widely employed in multiclass classification scenarios, the Soft-Max Classifier computes the probability distribution across all classes.

With an accuracy of 37.40 percent, the SVM Classifier employs Support Vector Machine techniques, particularly effective for binary and multiclass classification tasks, by determining optimal hyperplanes between classes.

The CNN Classifier, achieving the highest accuracy of 85.97 percent, relies on Convolutional Neural Network architecture tailored for image classification tasks. Through hierarchical feature extraction facilitated by convolutional layers, CNNs excel in discerning intricate patterns within images. These findings are summarized in Table 9.

Table 9 : The accuracy achieved by different classification algorithms

Classification Algorithm	Accuracy
K-NN Classifier	28.20%
Soft-Max Classifier	34.10%
SVM Classifier	37.40%
CNN Classifier	85.97%

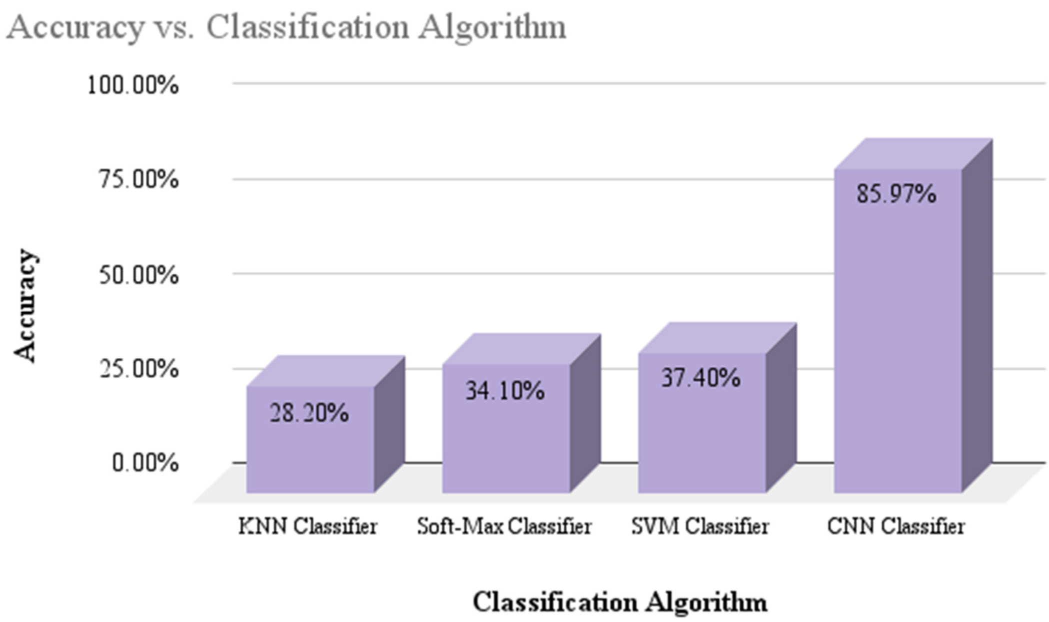

Fig 49 : The accuracy achieved by various classification techniques for current methodology

As mentioned in Table 10, The current BiLSTM achieved an F-measure of 91.23314. BiLSTM (Bidirectional Long Short-Term Memory) is a recurrent neural network architecture that is capable of capturing long-term dependencies in sequential data from both forward and backward directions. So for RNN it achieved an F-measure of 89.22156. RNN (Recurrent Neural Network) is a type of neural network architecture commonly used for sequential data processing tasks. For current DBN achieved an F-measure of 85.78629. The Deep Belief Network (DBN) is a generative neural network model composed of multiple layers of stochastic and latent variables. The Convolutional Neural Network (CNN) achieved an F-measure of 84.09091. CNN, short for Convolutional Neural Network, is a sophisticated deep learning architecture specifically designed for processing structured grid data, such as images.

Table 10 : The table shows the F-measure achieved by different algorithms

Algorithms	FMeasure
Existing BiLSTM	91.23314
Existing RNN	89.22156
Existing DBN	85.78629
Existing CNN	84.09091

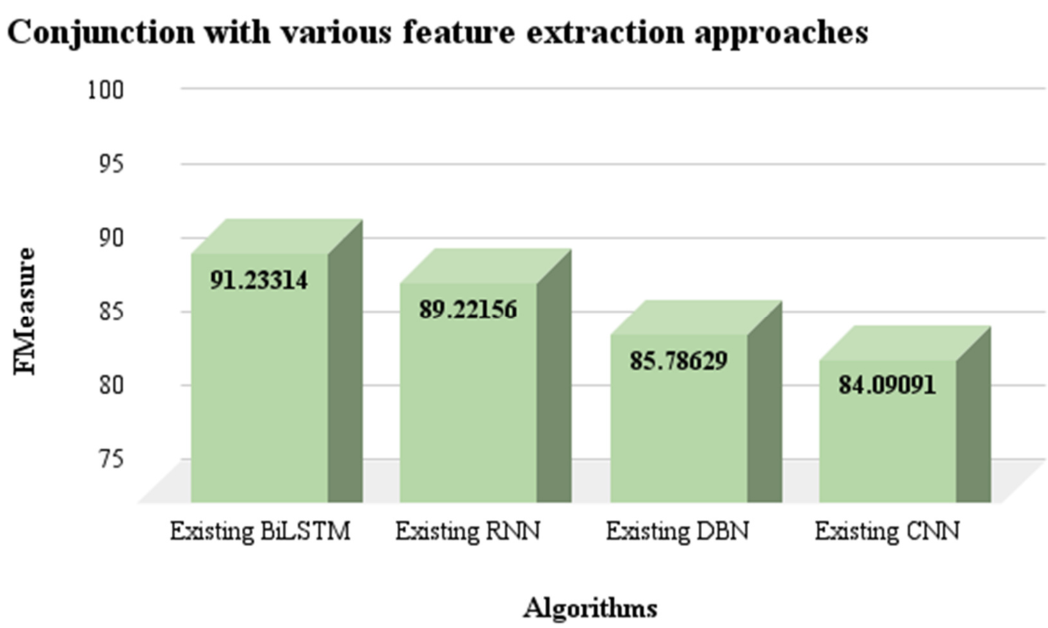

Fig 50 : F-measure achieved by different algorithms

Table 11 highlights the comparison of different algorithms which exist and with proposed algorithms. The convolutional neural network (CNN), dense based network (DBN), recurrent neural network (RNN) and Bidirectional LSTM are the current algorithms which are used for analysis. In which the highest F Measure is for the BiLSTM model. But when it is compared with the proposed Bidirectional LSTM the accuracy is high which is at 96.9%. So the proposed system predicts accurate output. This leads to the diagnostic system being very high quality.

Table 11 : The table compares the accuracy of both Proposed and Current algorithms

Algorithm	Accuracy
Proposed BiLSTM	96.9
Existing BiLSTM	91.33333
Existing RNN	89.1946
Existing DBN	85.9
Existing CNN	83.9

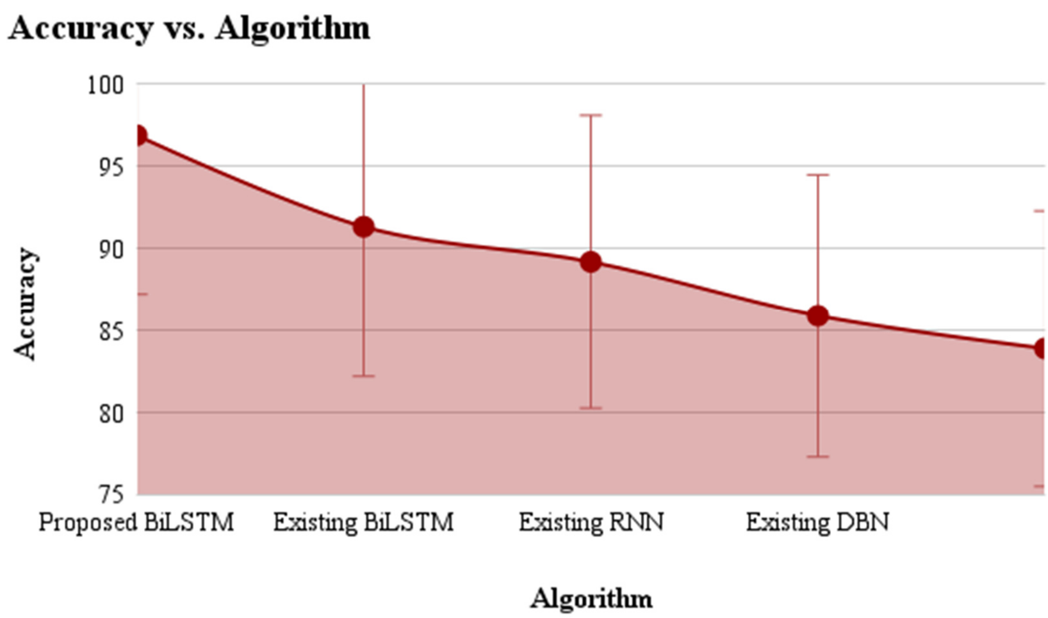

Fig 51 : Comparison with existing algorithm and proposed algorithm

Table 12 : The comparative analysis between Existing CNN and Existing BiLSTM with Proposed BiLSTM

Algorithm	Accuracy
Proposed BiLSTM	96.9
Existing BiLSTM	91.23
Existing CNN	85.97

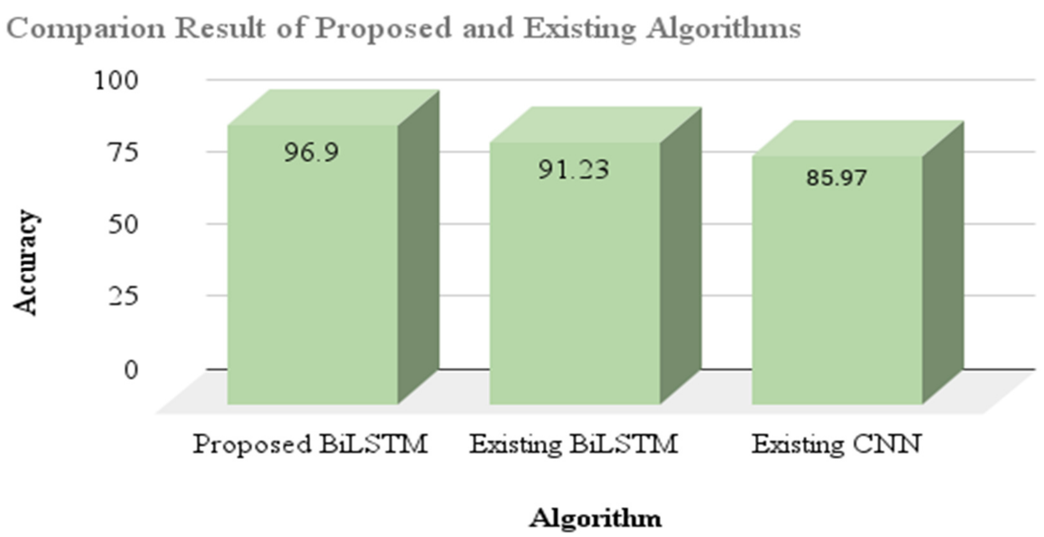

Fig 52 : The Comparison Result of Proposed and Existing algorithms

4.4 Snapshot on demonstration

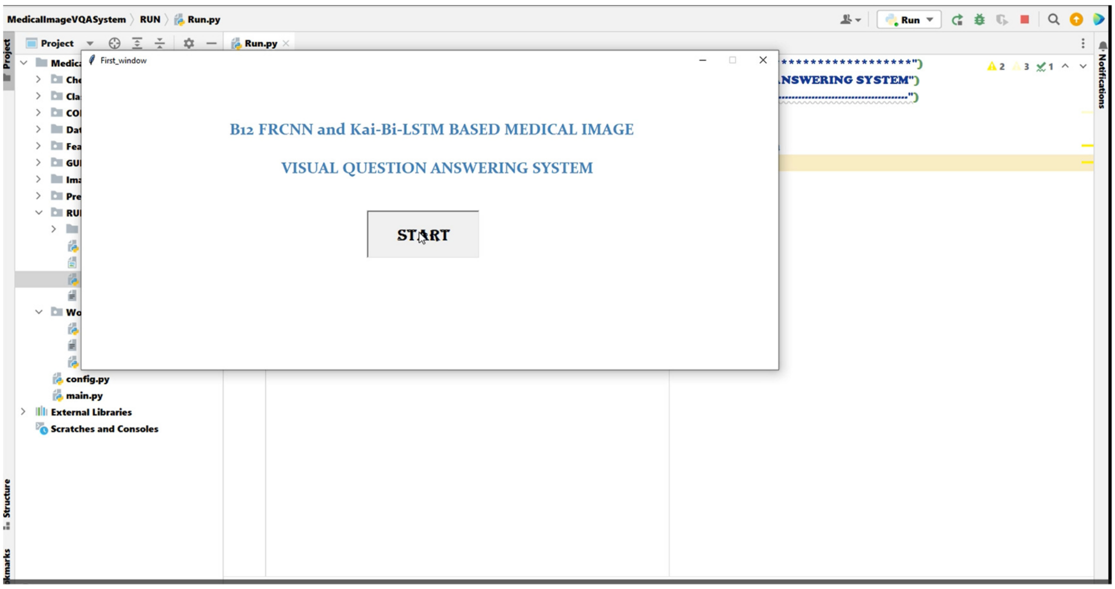

Fig 53 : Starting page of Application

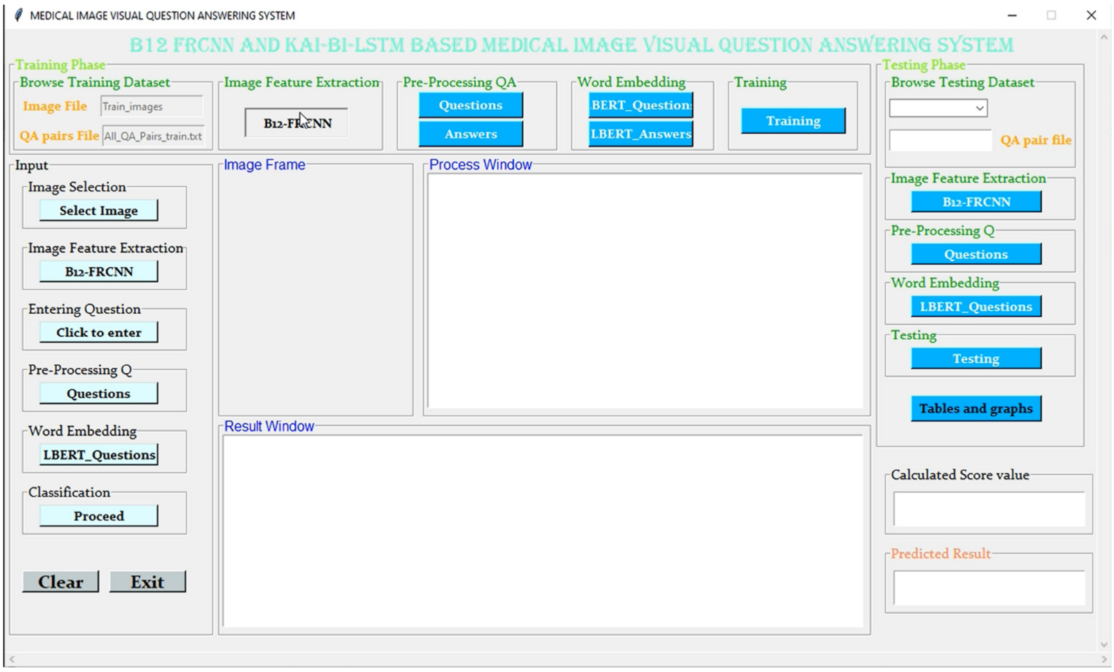

Fig 54 : Load the Training dataset both Image and Question Answer pair

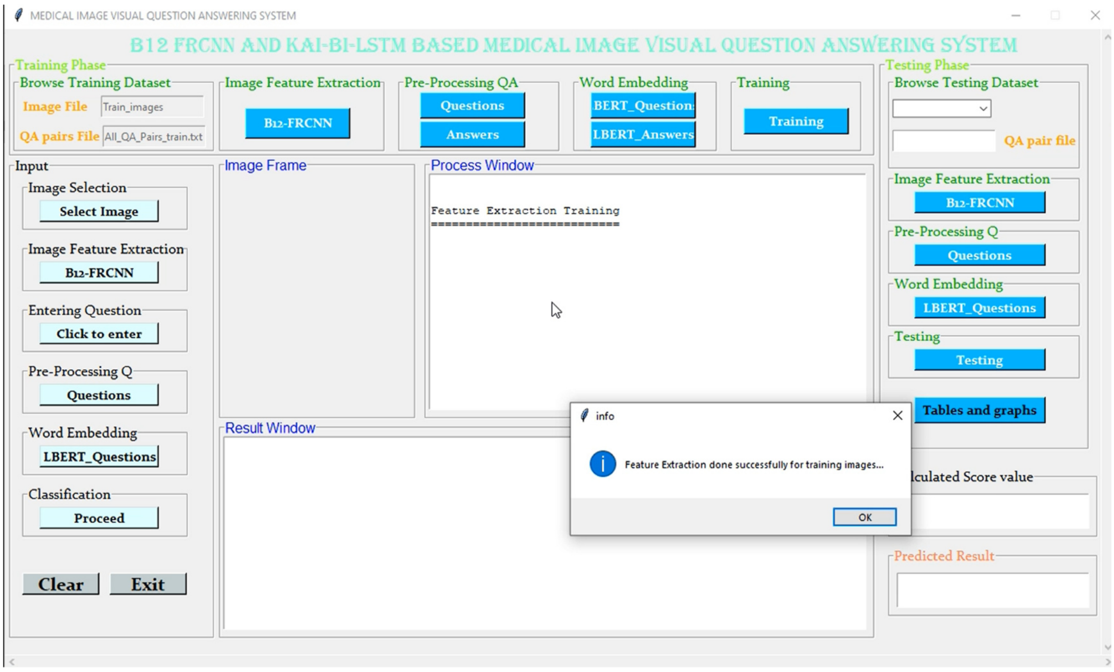

Fig 55 : Feature Extraction for Image dataset

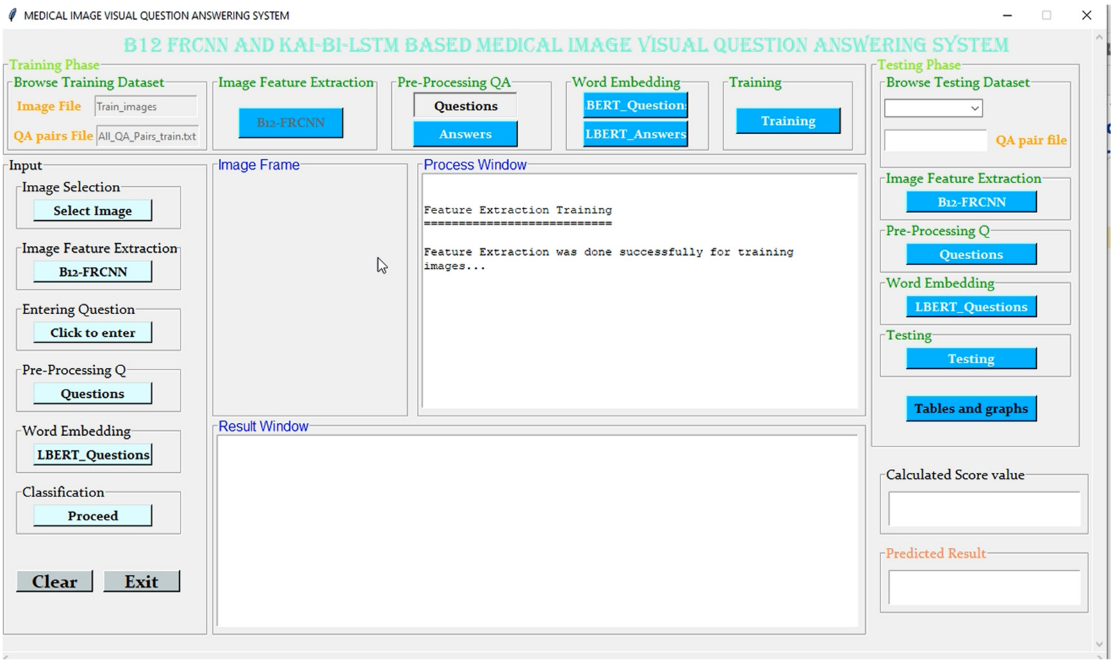

Fig 56 : Pre-process the Question dataset

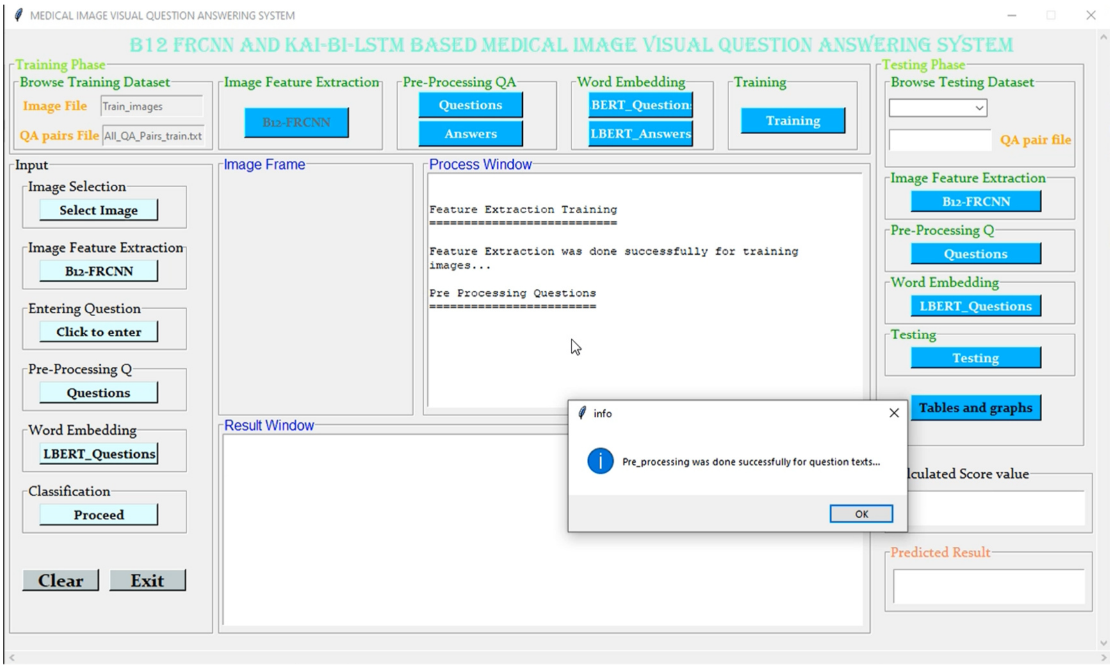

Fig 57 : Pre-process for Answer Dataset

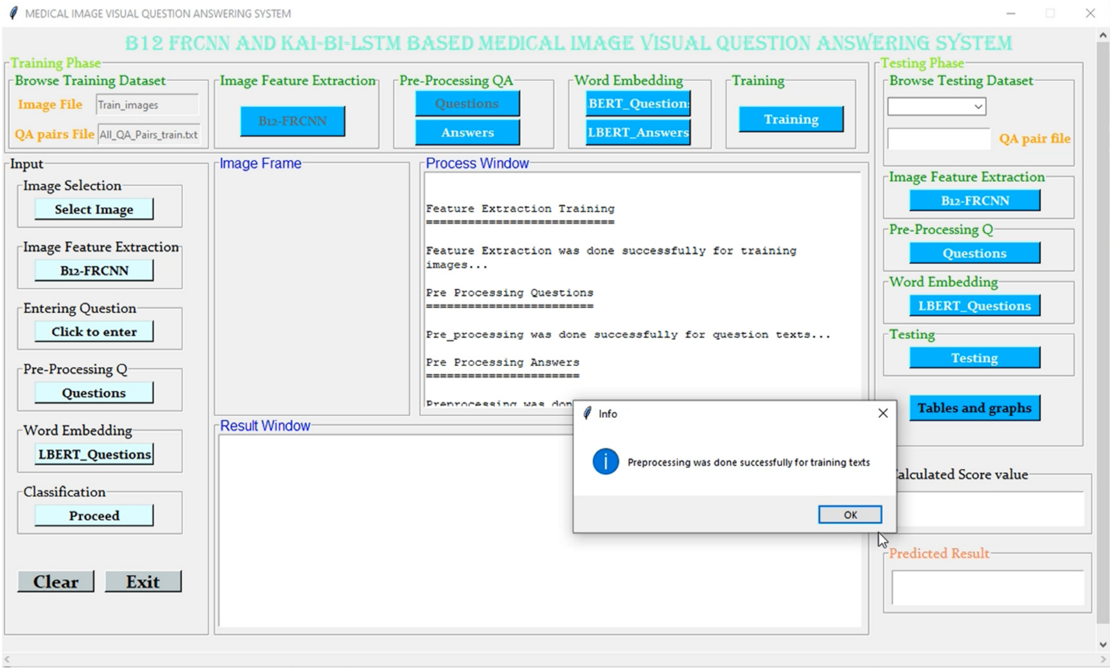

Fig 58 : Done with Preprocessing for both Question and Answer dataset

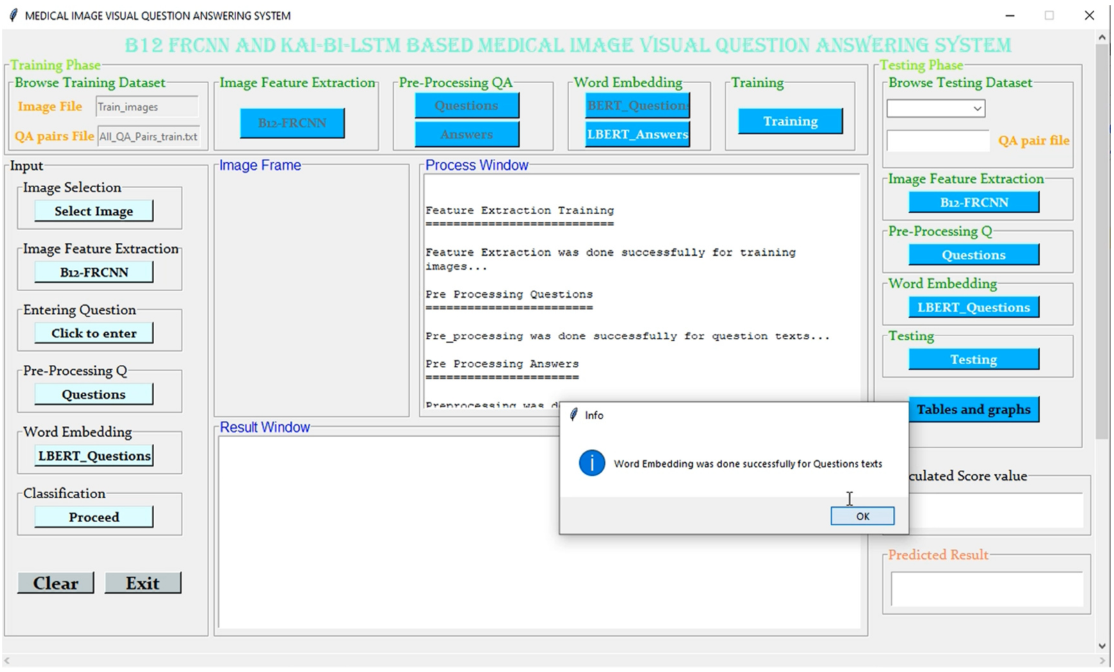

Fig 59 : Word Embedding for Question Dataset using BERT model

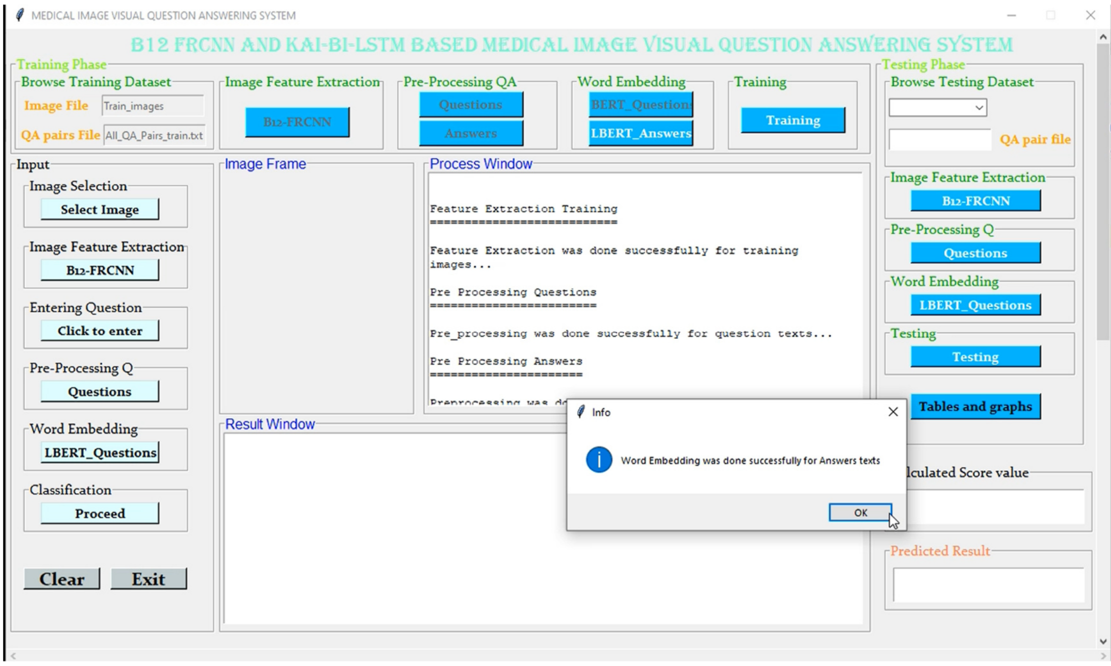

Fig 60 : Word Embedding for Answer Dataset using LBERT model

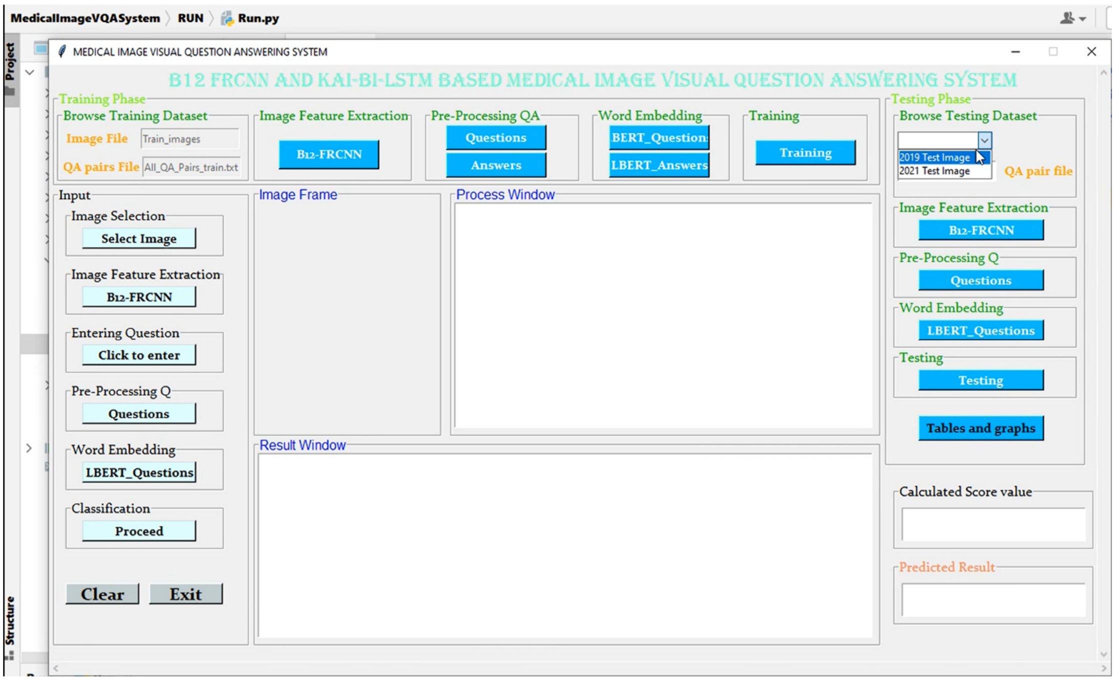

Fig 61 : Training the dataset for both visual and textual dataset

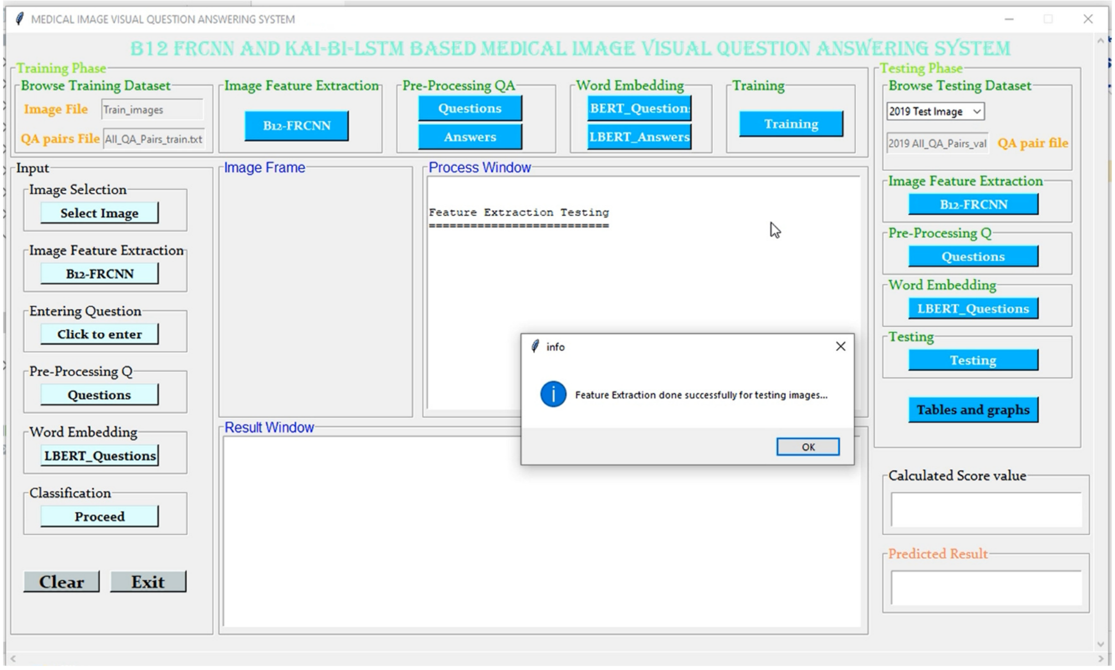

Fig 62 : Load the Testing dataset both Image and Question Answer pair

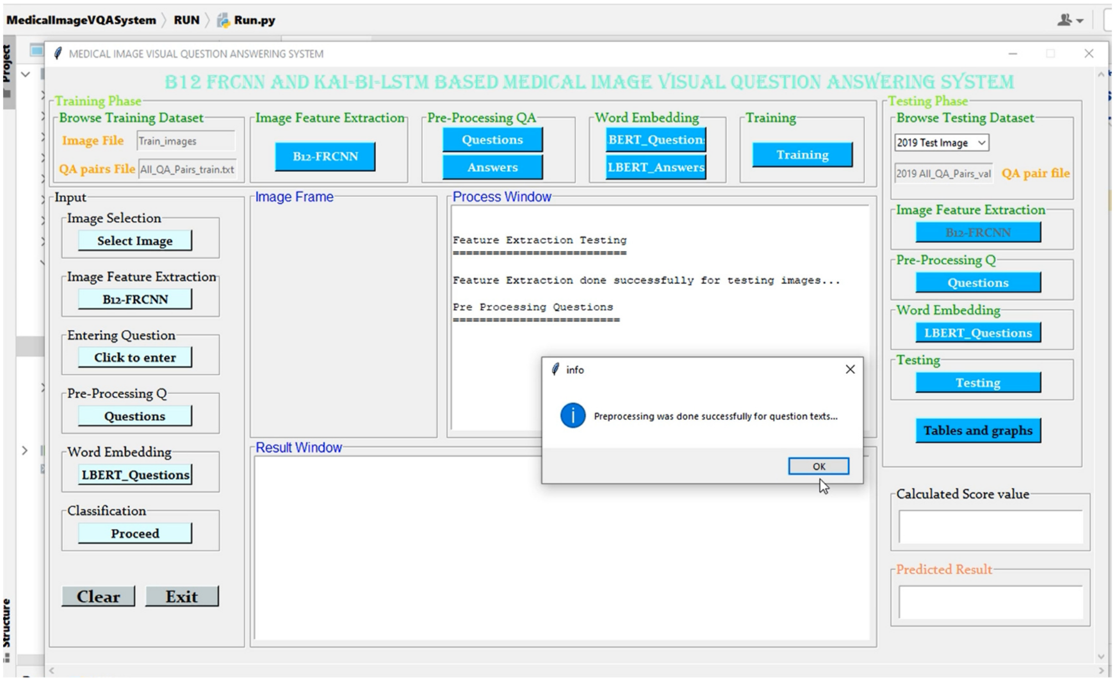

Fig 63 : Feature Extraction for Testing image dataset

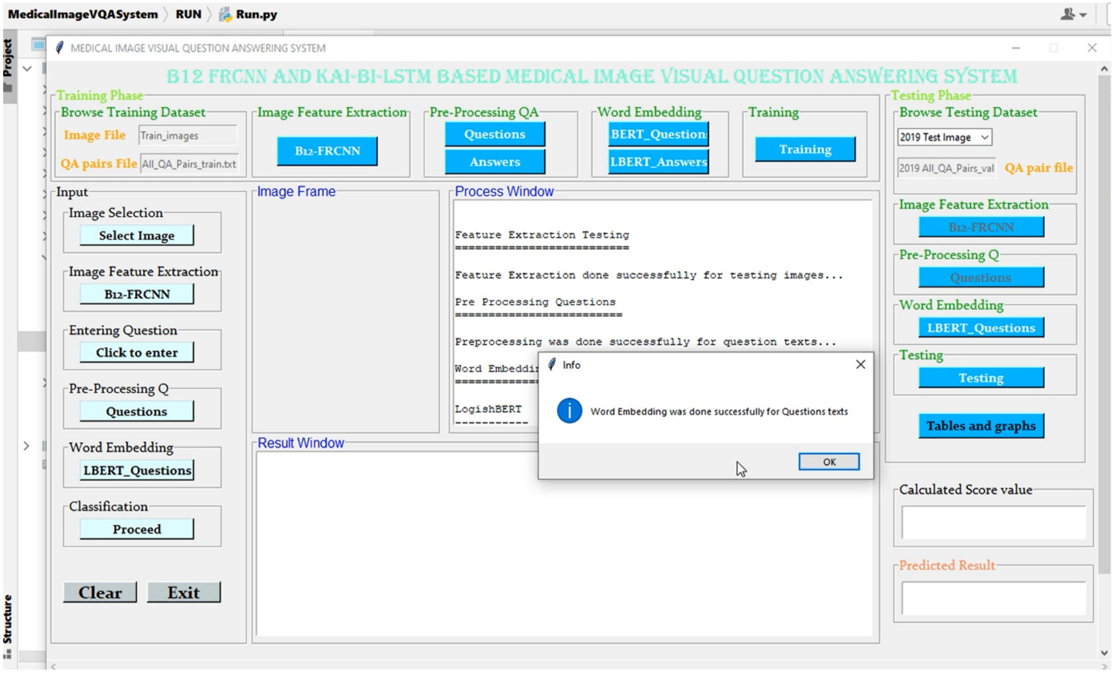

Fig 64 : Pre-processing for Question Datasets using LBERT model

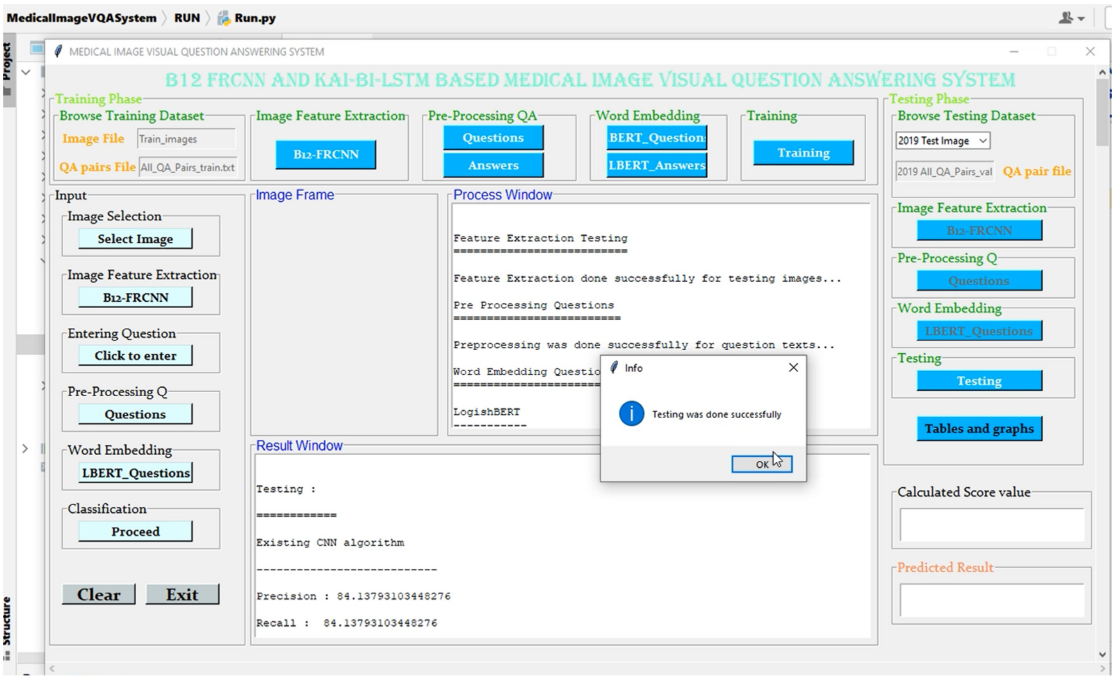

Fig 65 : Testing both visual and textual dataset

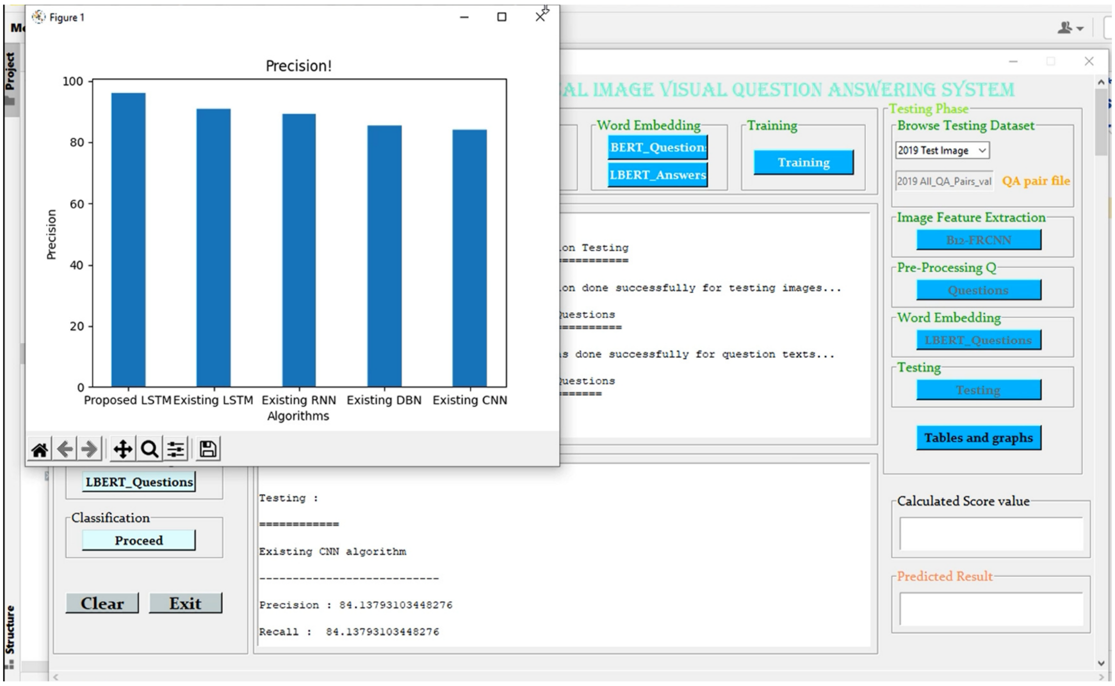

Fig 66- : Precision measure for proposed and existing models

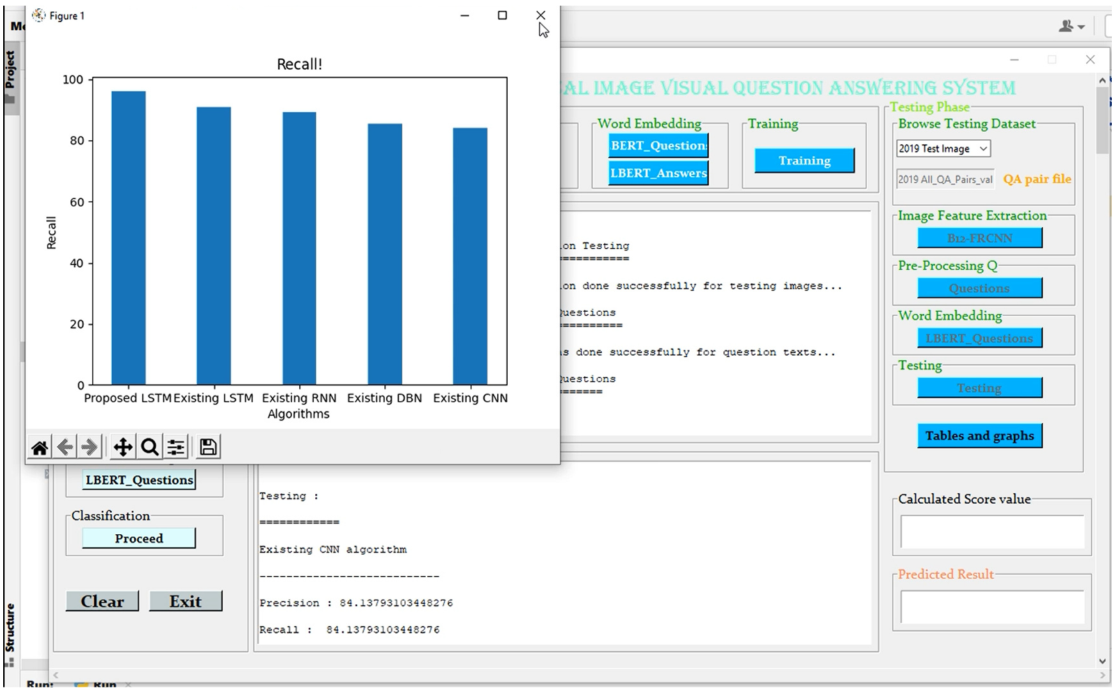

Fig 67 : Recall measure for proposed and existing models

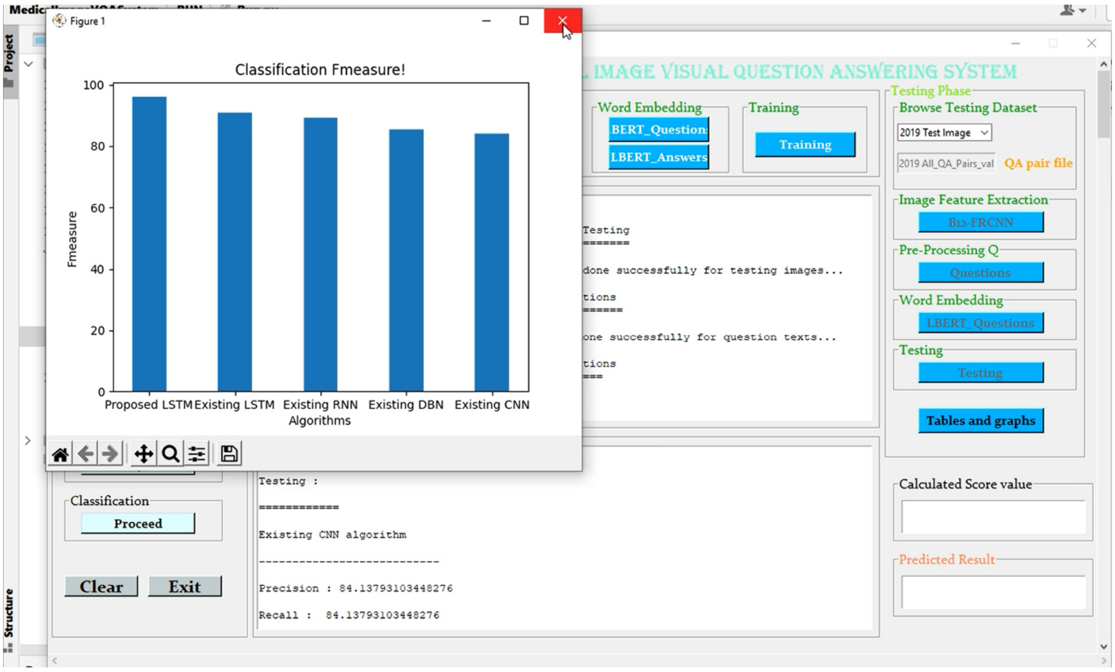

Fig 68 : Classification of FMeasure for proposed and existing models

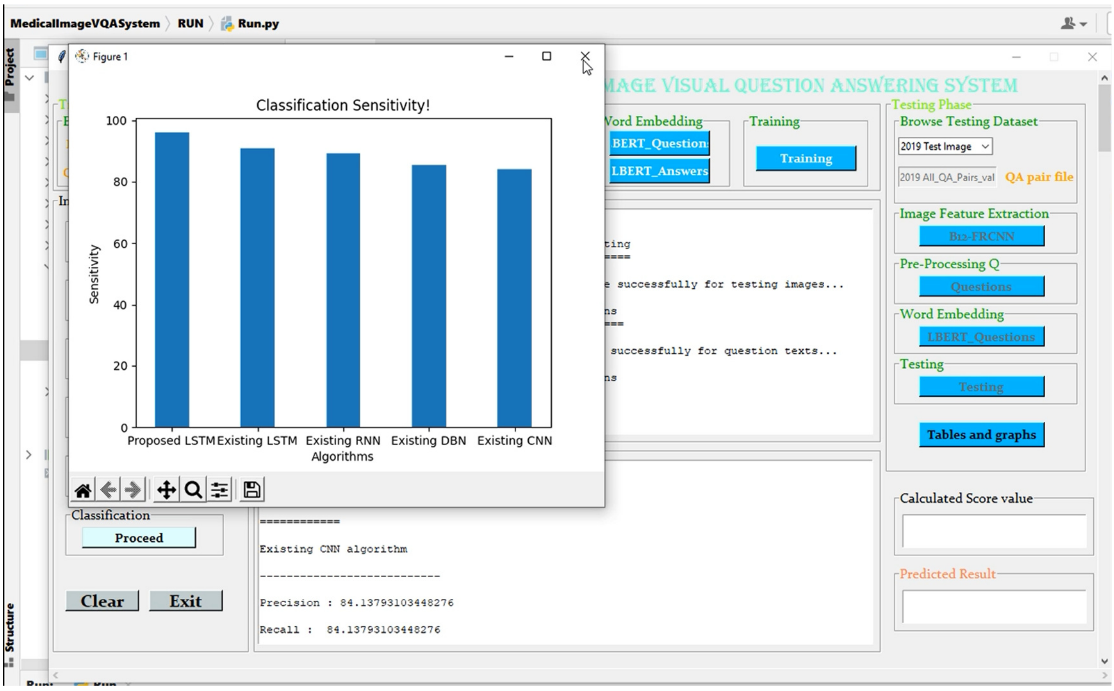

Fig 69 : Classification Sensitivity measure for proposed and existing models

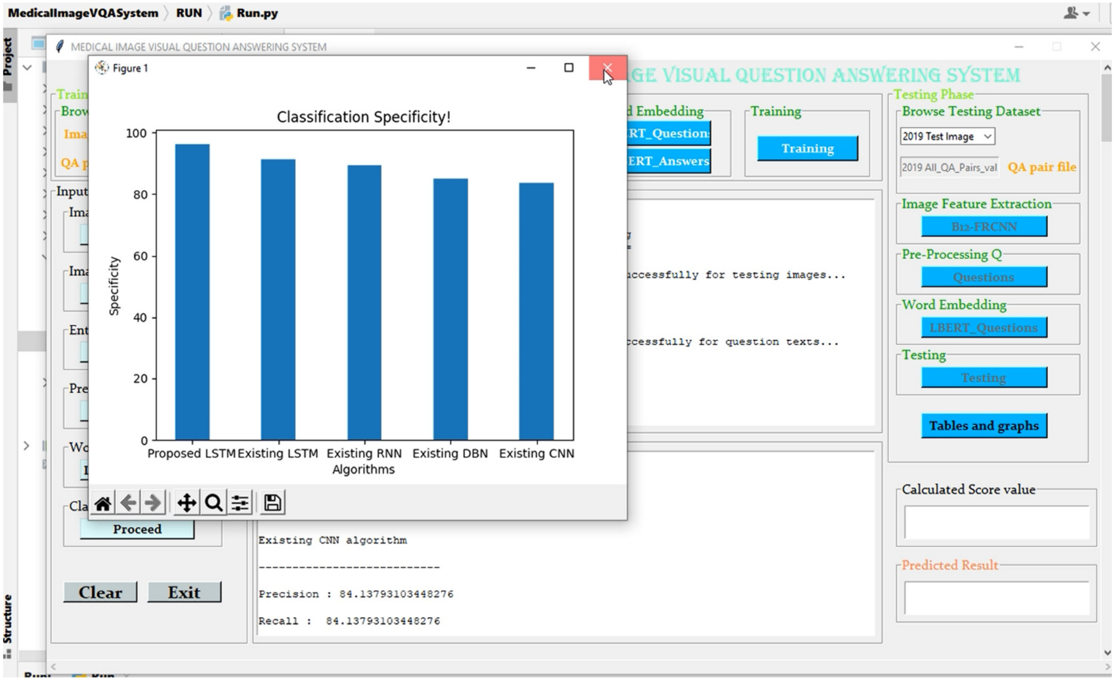

Fig 70 : Classification Specificity measure for proposed and existing models

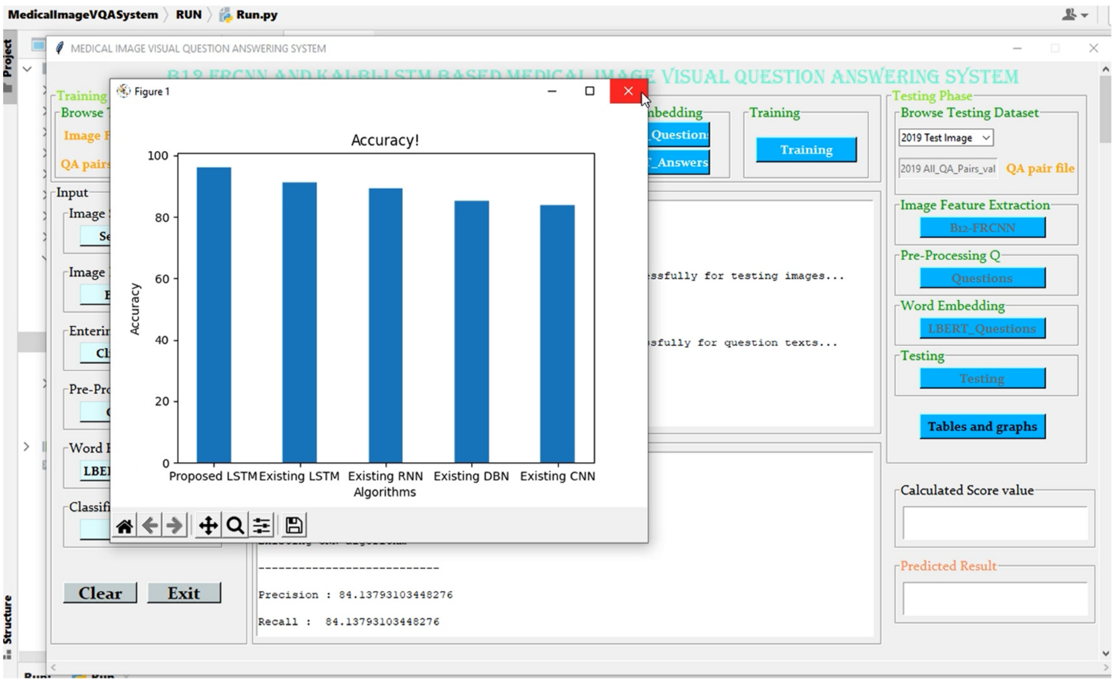

Fig 71 : True Positive Rate (TPR) measure for proposed and existing models

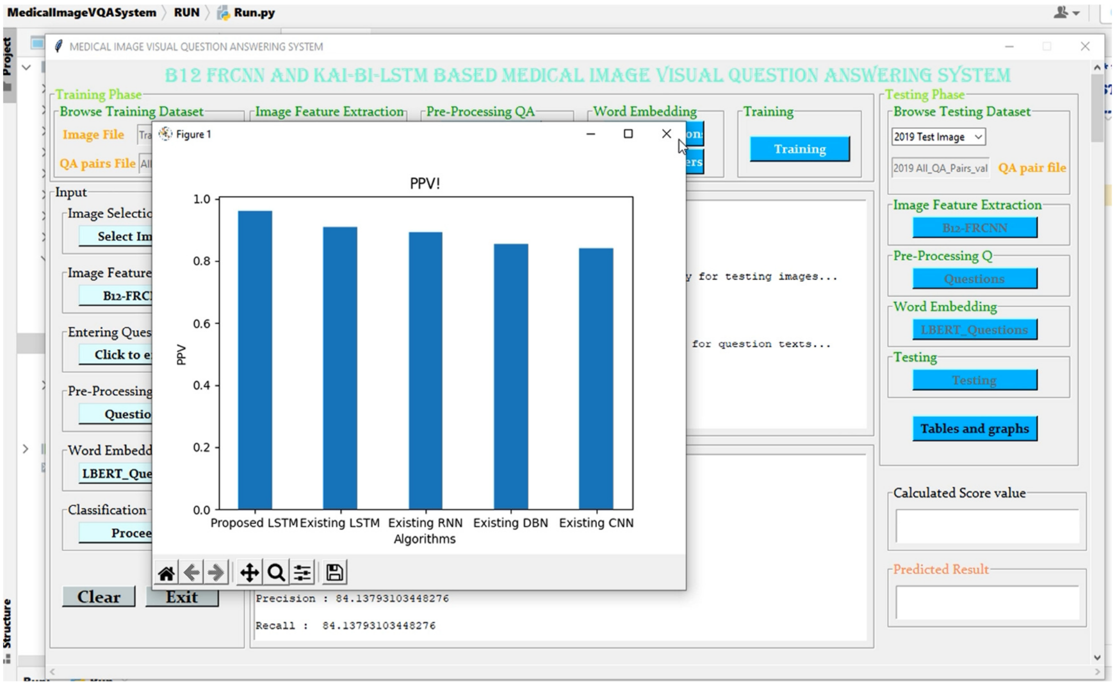

Fig 72 : Positive Predictive Value (PPV) measure for proposed and existing models

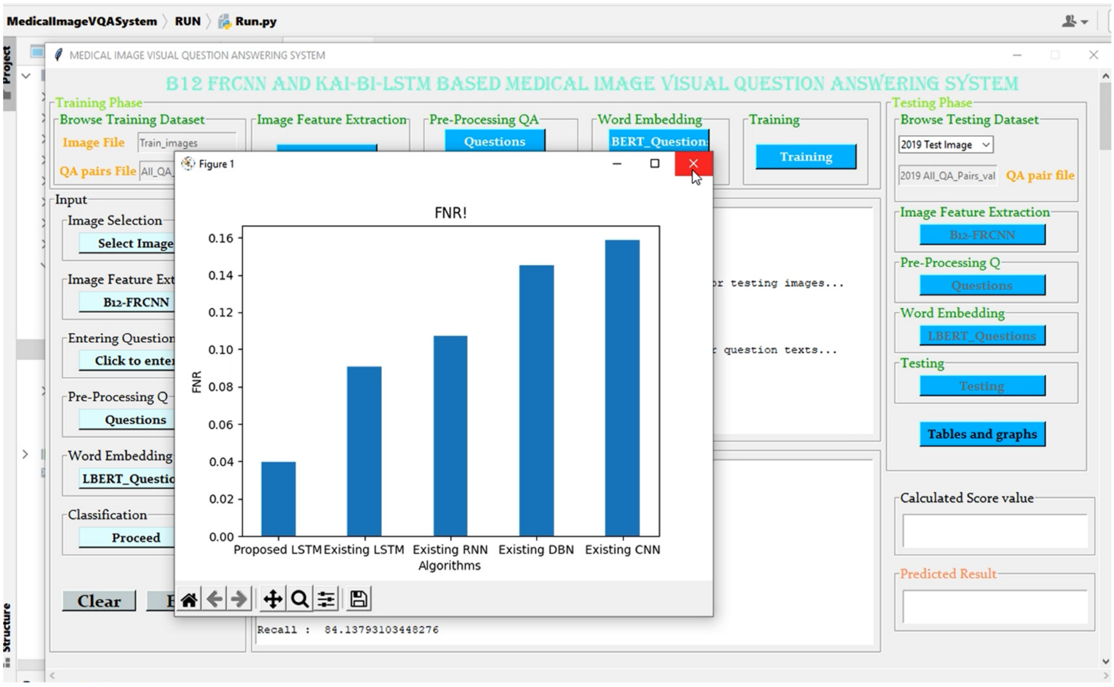

Fig 73 : Measure False Negative Rate (FNR) for proposed and existing models

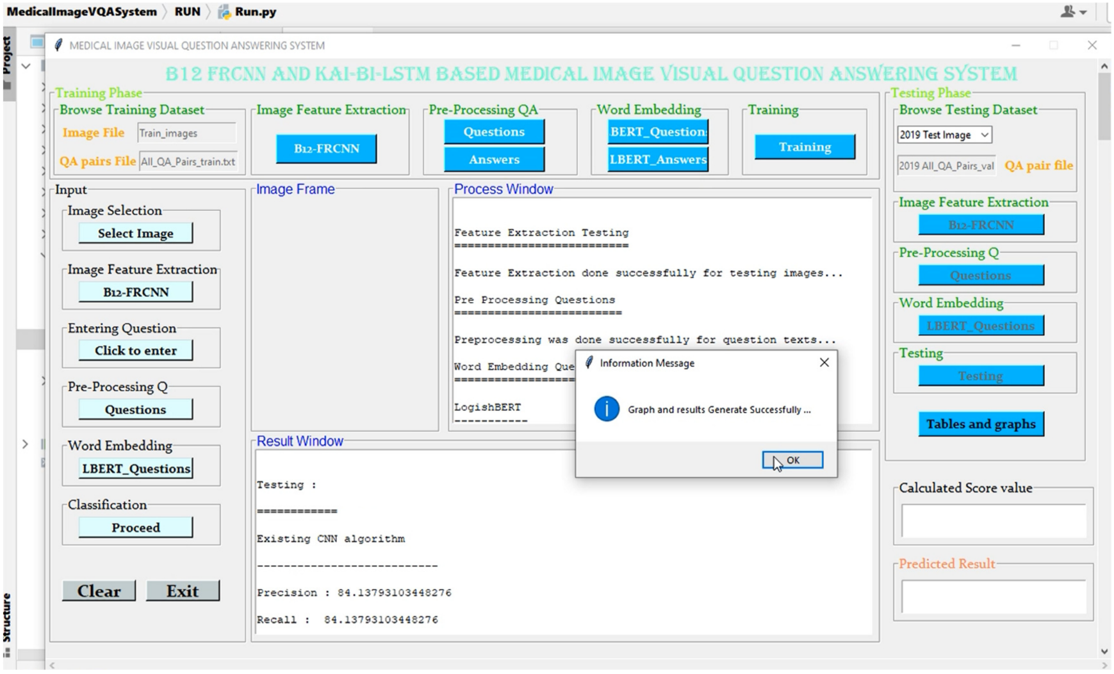

Fig 74 : To display Graphs and Tables

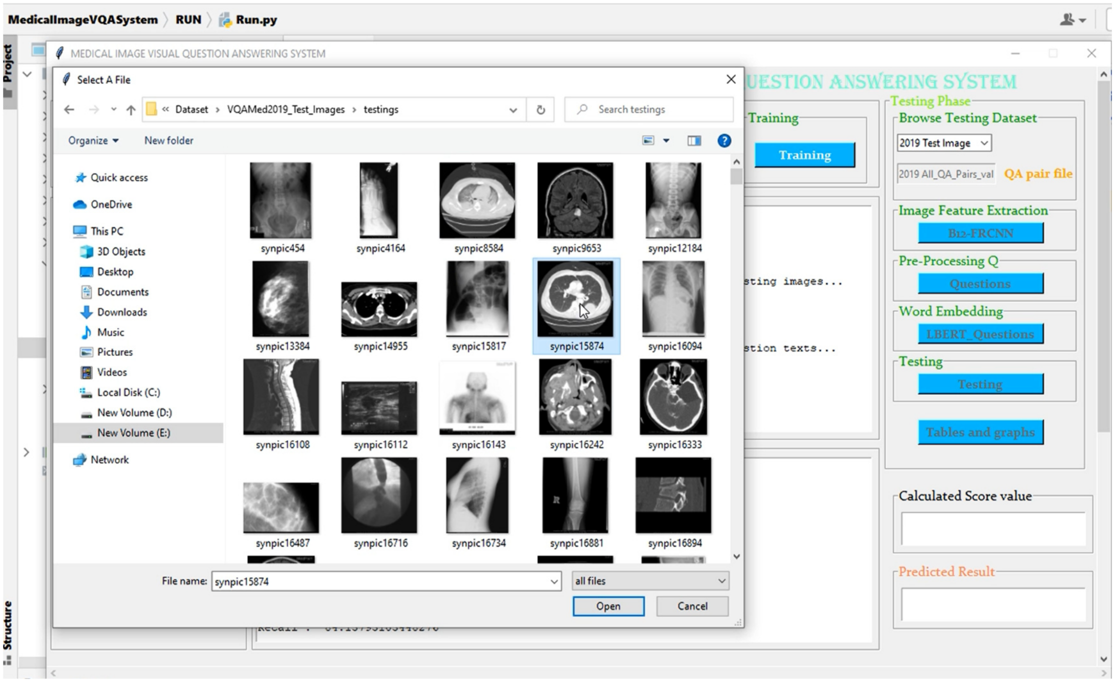

Fig 75 : Load the skeletal Image

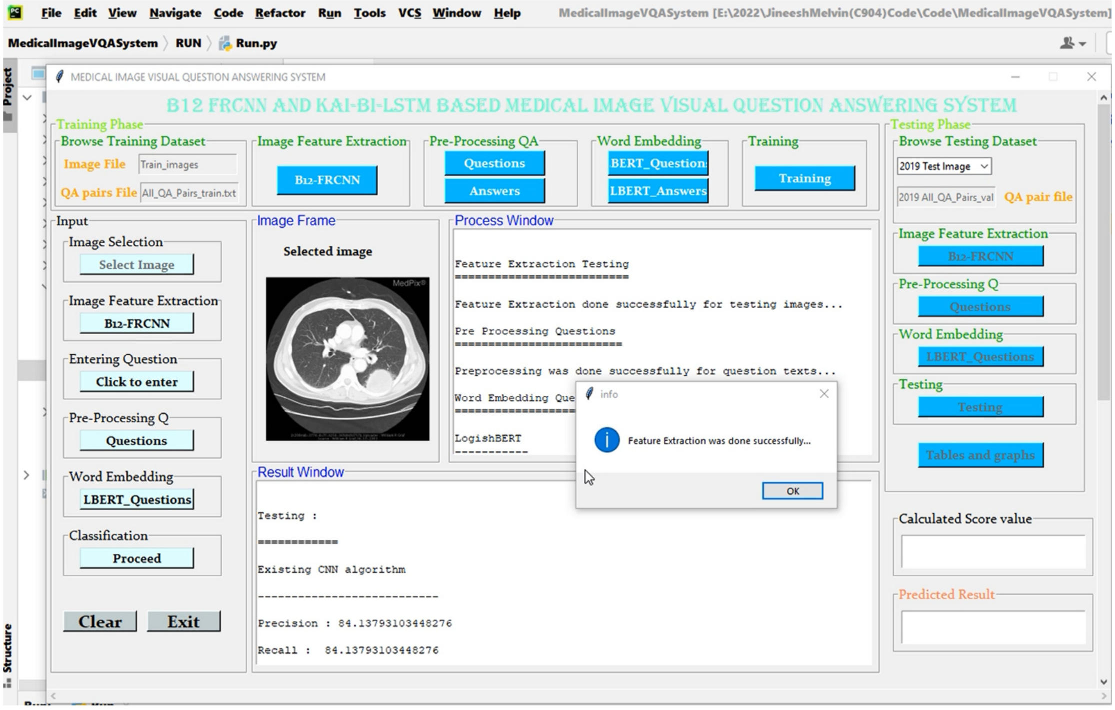

Fig 76 : Feature Extraction for Loaded Image using B12-FRCNN

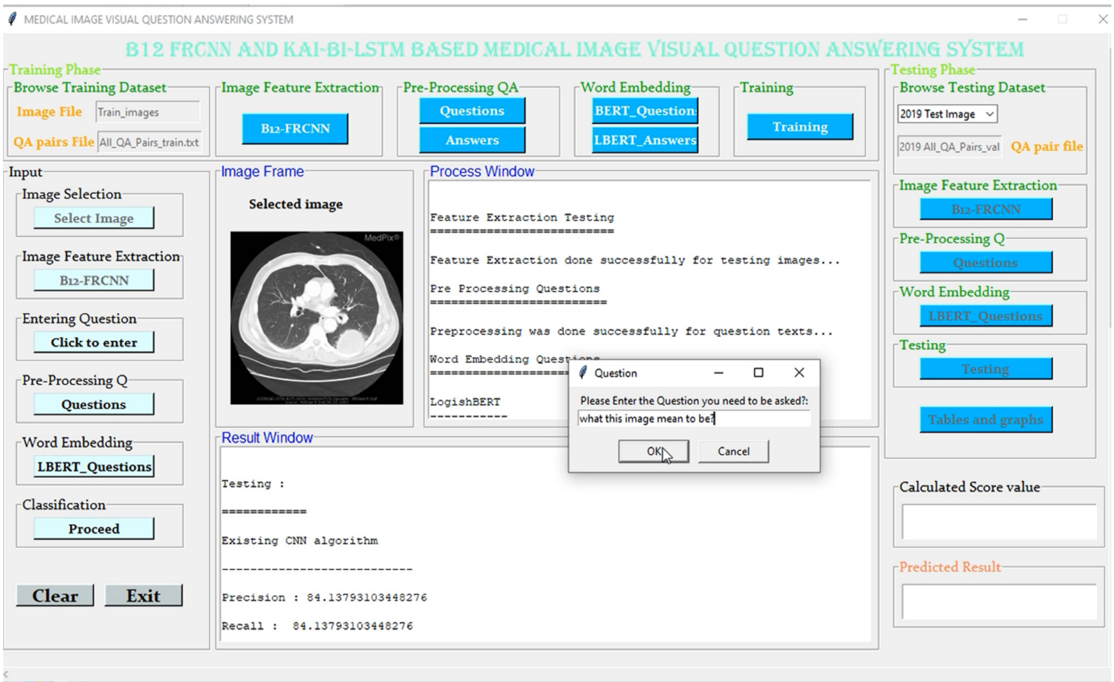

Fig 77 : Users Input Question

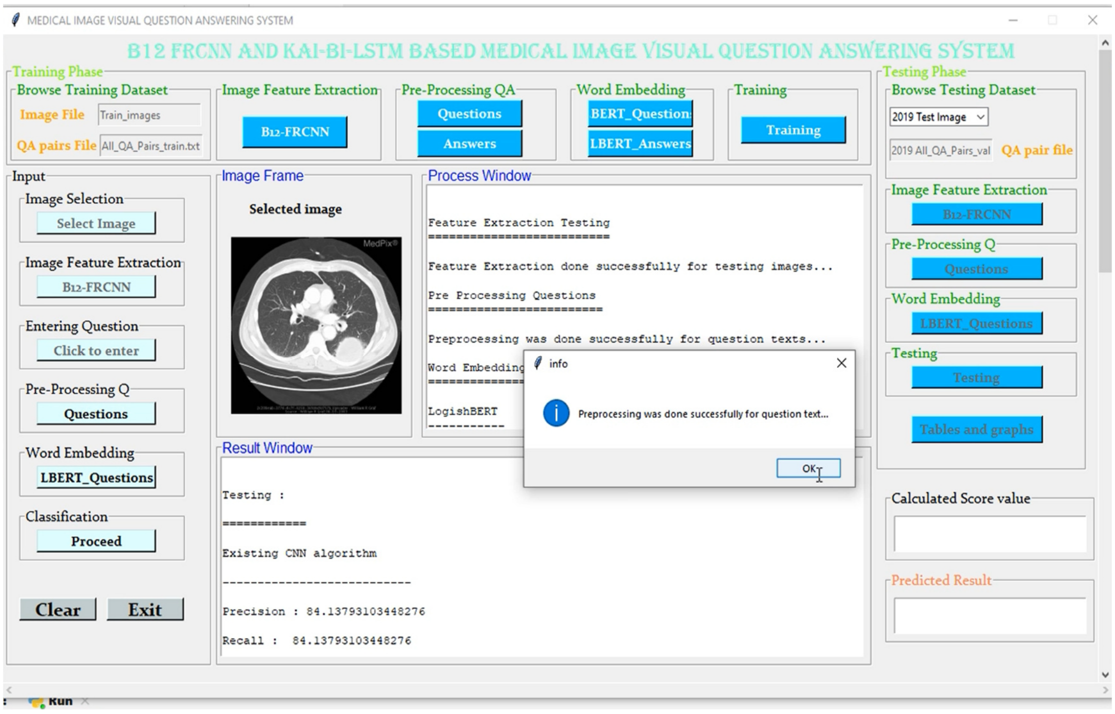

Fig 78 : Preprocess the Input Question

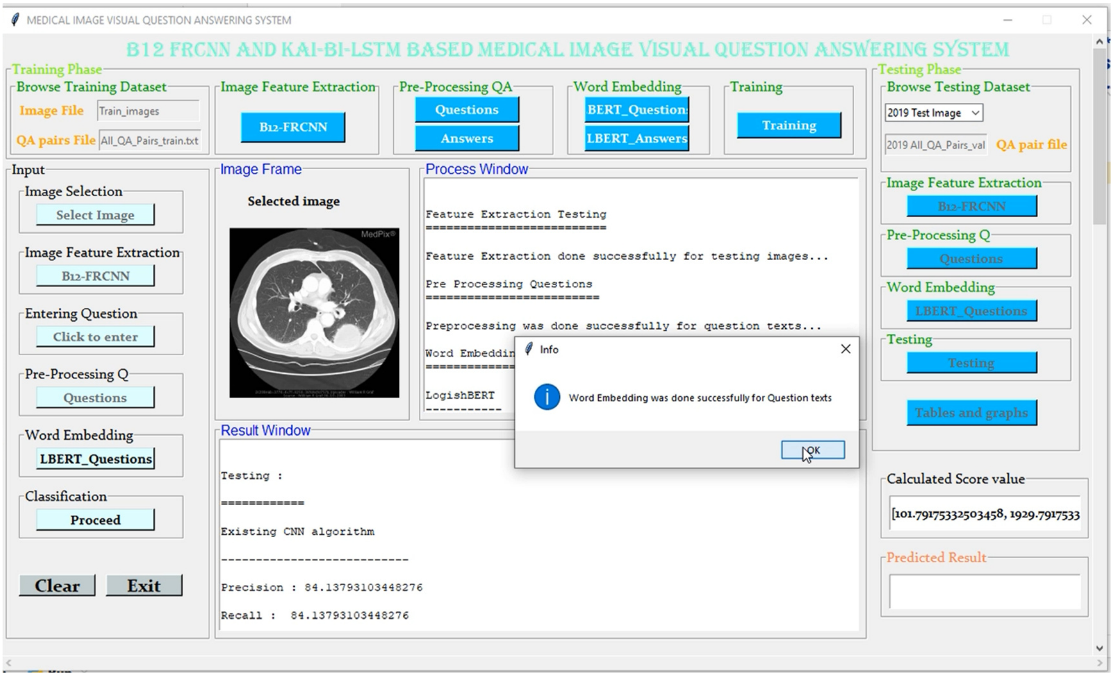

Fig 79 : Word Embedding for Input Question

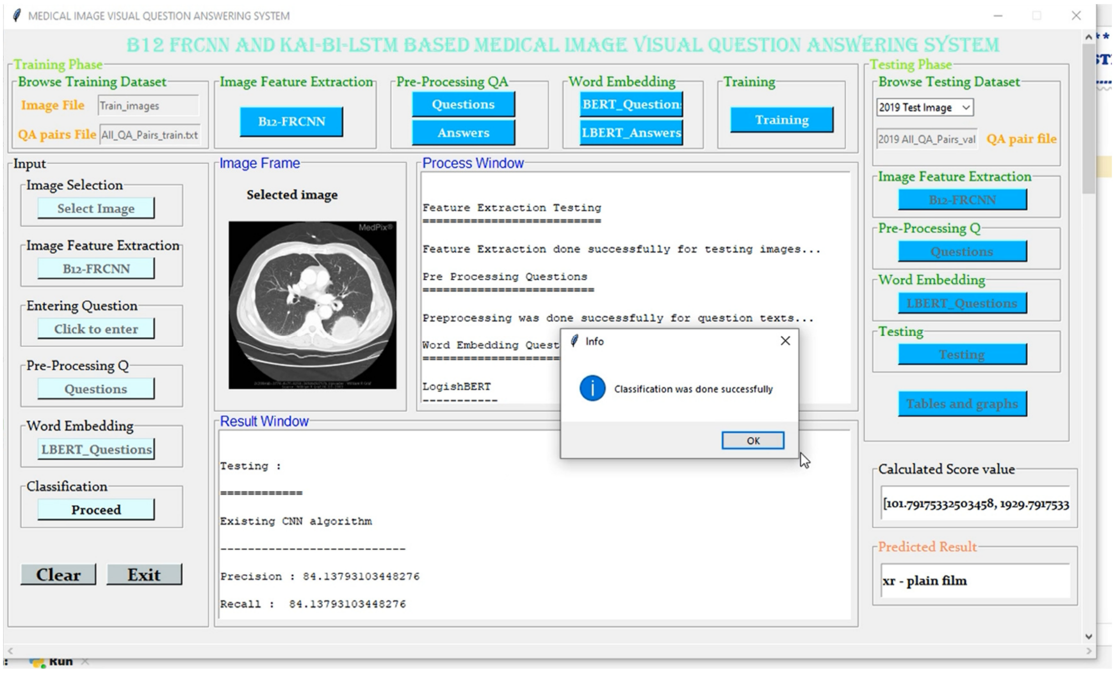

Fig 80 : Classification using Kai-BiLSTM model

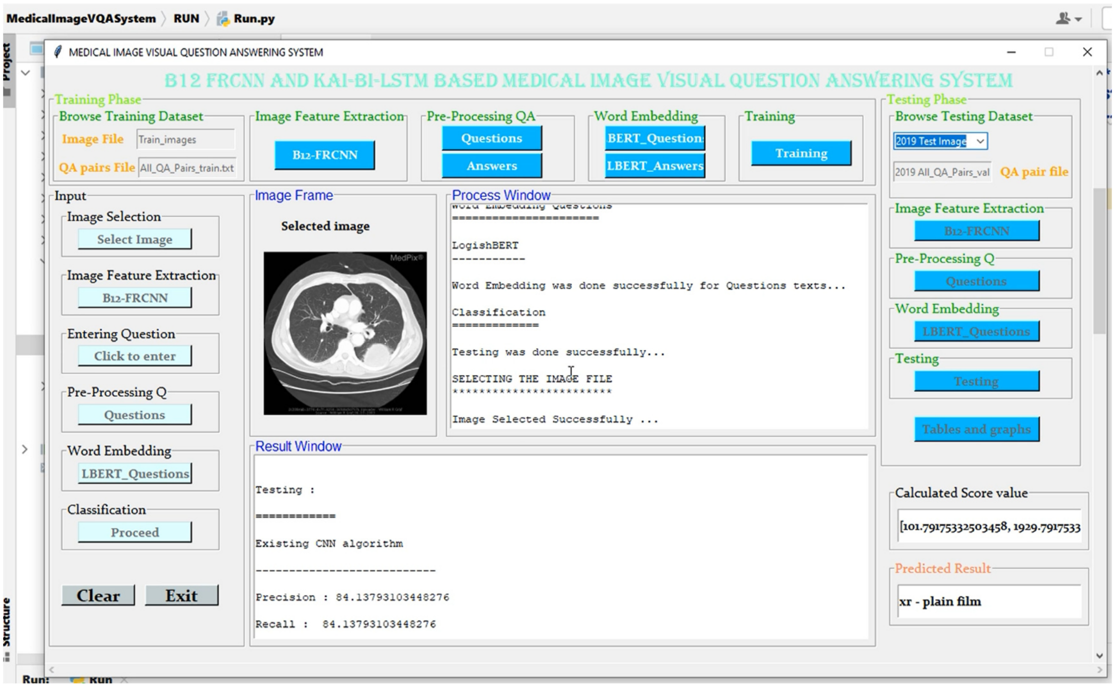

Fig 81 : Predicated answer with calculated score value for the answer