Predicting Severity Of Osteoarthritis Using Recurrent Convolutional Neural Networks (Rcnn) And Medical Imaging Data

B.Ashalatha[1], Thangarasan T[2], M.Geetha[3], Saranya T[4], Manisha Mittal[5], VenkataRamana K[6]

[1]*Associate Professor, Department of CSE, Andhra Loyola Institute of Engineering and Technology, Vijayawada, Andhra Pradesh, India. latha009asha@aliet.ac.in*

[2]*Assistant Professor, Department of Computer Science & Engineering, Madanapalle Institute of Technology & Science, Madanapalle. thangarasant@mits.ac.in*

[3]*Professor, Department of IT, S.A.Engineering College, Poonamallee-Avadi Road, Thiruverkadu, Chennai, Tamilnadu, India. geetham@saec.ac.in*

[4]*Assistant Professor III, Computer Science and Engineering, Velammal College of Engineering and Technology, Madurai, Tamilnadu, India. asv@vcet.ac.in*

[5]*Department of Electronics and Communication Engineering, Guru Tegh Bahadur Institute of Technology, GGSIPU, New Delhi, India. manumanisha22@gmail.com*

[6]*Associate Professor, Department of CSE, Sri Vasavi Engineering College, Pedatadepalli, Tadepalligudem, Andrapradesh, India. kvrbook@gmail.com*

KEYWORDS

Osteoarthritis, Recurrent Convolutional Neural Networks, Medical Imaging, X-ray Classification, Severity Grading

ABSTRACT

Cartilage degeneration causing functional incapacity and discomfort defines common joint condition Osteoarthritis (OA). Correct classification of OA severity from knee X-ray images determines diagnosis and treatment course. Conventional methods of OA severity classification rely on hand inspection or basic machine learning techniques, which could not effectively capture the complex trends in imaging data. Using a dataset of 9,786 knee X-ray images, this work uses Recurrent Convolutional Neural Networks (RCNN) to project OA severity. Combining recurrent layers in the RCNN architecture helps to capture temporal correlations in spatial information, hence enhancing classification performance. Defining the dataset are five KL grades: 0 (healthy) to 4 (severe). This beats by around 5%, 7%, and 3%, respectively standard CNN, Deep Neural Network (DNN), and Deep Convolutional Neural Network (DCNN), with a precision of 87.2%, recall of 88.9%, accuracy of 89.5%, and F1-score of 88.0%.

1. Introduction

Knee osteoarthritis (OA) is a common degenerative joint disease typified by progressive degradation of the articular cartilage, which cushions and protects the bones in the knee joint [1]. When the cartilage breaks down, bones start to rub together and cause discomfort, stiffness, and limited movement [2]. Moreover impacting the underlying bone and surrounding soft tissues is the condition [3]. Often called knee arthritis, OA is the most common kind of arthritis affecting the knee and can adversely affect a person's quality of life [4]. Early initial diagnosis of OA determines effective therapy and management [5]. From 0 (healthy) to 4 (severe—medical imaging, especially X-rays, is vital in assessing the degree of OA [using the Kellgren-Lawrence (KL) grading system splits the illness into five grades].

The present challenge is to use sophisticated machine learning methods to create an automated system employing X-ray pictures estimating the degree of knee osteoarthritis specifically, the aim is to use deep learning models that can learn complex information from photos and generate consistent predictions to increase the accuracy and efficiency of OA degree of classification. Integrating convolutional and recurrent neural networks to properly manage the sequential character of features and the unpredictable imaging data is a difficulty.

The primary objectives of this research are:

1. To create an automatic OA severity classification recurrent convolutional neural network (RCNN) model knee X-ray image.
2. To evaluate the performance of the RCNN model.
3. To analyse the model will aid to evaluate its ability to raise by means of varying dataset sizes.
4. To investigate the model's performance under several degrees of validation, testing, and training data holistically.

The novelty of this research is the improvement of feature extracting and classification by use of recurrent layers mixed with convolutional layers. Conventional CNNs can find sequential dependencies and contextual information challenging even if they are good in capturing spatial features. By incorporating recurrent layers, the RCNN model aims to identify temporal links and sequential patterns in the data obtained from X-ray images, therefore enabling improved performance in assessing the degree of osteoarthritis.

This research contributes to the field by:

1. The research develops novel RCNN architecture combining the strengths of convolutional and recurrent layers to provide better feature learning and classification.
2. The research shows the benefit in handling varied and difficult datasets by providing a complete evaluation of the performance of the RCNN model against more conventional methods.
3. The authors provide knowledge of how dataset size influences model performance, hence directing data collecting and model training in applications of medical imaging.
4. Both doctors and patients will benefit from automated diagnostics technologies able to improve OA degree of accuracy and efficiency.

Related Works

In medical imaging tasks, including the convolutional layers, CNNs automatically extract hierarchical features from images, therefore enabling the identification of patterns and structures relevant for classification tasks. Using Kellgren-Lawrence grading system to estimate the degree of osteoarthritis using CNNs to analyse knee X-ray images, one well-known application of CNNs is osteoarthritis identification in [11]. Their method shown that CNNs could effectively learn and categorise image characteristics, so attaining great accuracy in diverse OA gradients. CNNs may thus be constrained in challenging classification issues since they are basically geared to control spatial features and may not effectively capture sequential dependencies or contextual information.

Deep Neural Networks (DNNs) extend standard neural network models by include additional hidden layers to capture more complex patterns and connections in data. Medical imaging has seen DNNs applied for a range of diagnostic tasks including disease categorisation and prediction. For instance, [12] looked at knee MRI data searching for osteoarthritis using DNNs. Their method was better than traditional methods, thereby proving DNNs' ability to mimic intricate patterns in imaging data. DNNs can thus represent complex features, but they also usually demand high computing resources and may suffer from overfitting without suitable regularising.

Deep Convolutional Neural Networks (DCNNs) outperform more traditional CNNs by including deeper architectures with additional convolutional and pooling layers. This depth increases image classification job accuracy and allows DCNNs gather more complicated details. Learning more abstract and high-level information from knee X-ray and MRI images has helped DCNNs to enhance classification performance in osteoarthritis. [13] classified osteoarthritis degree using a DCNN-based method and found appreciable accuracy gains over traditional CNN methods. Their model gathered contextual information and finer details—qualities quite necessary for proper OA grading—using deeper network architectures. DCNNs can be computationally demanding and need on large amounts of labelled data to obtain maximum performance, though.

Combining convolutional and recurrent layers, recurrent convolutional neural networks (RCNNs) help to alleviate CNN and DCNN restrictions in handling sequential and contextual data. RCNNs significantly help when sequential dependencies and context are very crucial for feature extraction and classification. Among the various image analysis uses for which current research have looked at RCNN use is medical imaging. For cancer detection and classification in medical images, for example, [14] proposed an RCNN-based model showing that recurrent layers could increase the model's capacity to record temporal and contextual linkages inside the images. By aggregating recurrent layers to boost the model's capacity to learn and classify difficult information from knee X-ray images, using RCNNs to detect osteoarthritis tries to optimise these benefits.

From CNNs to DCNNs and with the incorporation of RCNNs, image analysis and classification have made significant development. CNNs and DCNNs have demonstrated success in feature extraction and classification; RCNNs have an enhanced capacity by integrating recurrent layers, which may record sequential dependencies and contextual information even if they exhibit success in these aspects as well. Extending these advances, the suggested use of RCNNs for osteoarthritis severity classification aims to offer a more dependable and accurate approach for medical image analysis.

2. Methodology

In this section, combining convolutional and recurrent neural network components, the RCNN improves feature extraction and classification performance especially appropriate for medical imaging data where spatial and temporal information is critical as in Figure 1.

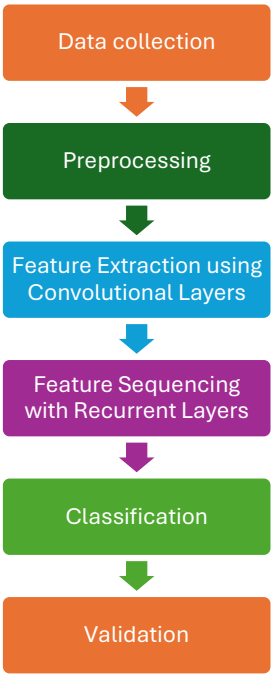

Figure 1: Proposed RCNN Framework

1. **Preprocessing**:

- Image Resizing: Resize all knee X-ray images.
- Normalization: Normalize pixel values to standardize input data.

2. **Feature Extraction using Convolutional Layers**:
   - Input Layer: Feed the preprocessed images into the RCNN.
   - Convolutional Layers: Each convolutional layer captures different aspects of the image.
   - Activation Function: Use Rectified Linear Unit (ReLU) activation function to introduce non-linearity.
   - Pooling Layers: Apply max pooling to reduce the dimensionality of feature maps while retaining essential features.
3. **Feature Sequencing with Recurrent Layers**:
   - Flattening: Flatten the output from the convolutional layers to create a sequence of feature vectors.
   - Recurrent Layer: Feed the flattened features into a Recurrent Neural Network (RNN) layer, typically an LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit),.
4. **Classification**:
   - Fully Connected Layers: Pass the output of the recurrent layer through one or more fully connected (dense) layers to map the extracted features to the final classification space.
   - Softmax Activation: Apply a softmax function in the output layer to get probability scores for each severity grade (0 to 4).
5. **Training**:
   - Loss Function: Use categorical labels.
   - Optimizer: Utilize the Adam optimizer to adjust weights during training.
   - Evaluation: Validate the model.
6. **Testing and Evaluation**:
   - Test Set: Evaluate the model's performance on a held-out test set.

Pseudocode
Proposed Algorithm:
Step 1: Preprocessing
Step 2: Feature Extraction on convolutional, Pooling
Step 3: Feature Sequencing based on Shapes, Length, LSTM
Step 4: Classification on 5 Classes with 128 unit's dense layer
Step 5: Training Model, epochs=50, batch size=32
Step 6: Testing and Evaluation

Pre-processing

In terms of determining the degree of osteoarthritis, pre-processing is a crucial initial step for producing knee X-ray images for machine learning models analysis. Part of this phase are several crucial operations aimed to assure the photos suit the format for the next phases of feature extraction and model training and increase their quality.

First, image scaling brings all input image dimensions into line. This collection's every image is reduced to 224 by 224 pixels. This homogeneity in size ensures that every image entered into the model has the same spatial dimensions, therefore preventing discrepancies that may otherwise jeopardise the

model's performance. Since constant size processing of images is more efficient, resizing also helps effectively manage computer resources.

Normalising following scale helps to change the image pixel values. Usually working with division by the highest feasible value—usually 255 for 8-bit images—normalization scales pixel values to a range of [0, 1]. This stage helps to assure that the model is not biassed towards higher-intensity values in the images and to improve the convergence speed of the learning technique. Normalised images most certainly emerge from a more consistent and accurate training process.

Apart from resizing and normalising, other preparation tasks, so providing the model variations of the training data and so boosting its generalising capacity. However, in this context scaling and normalising as basic preprocessing techniques take front stage.

Pre-processing therefore turns unprocessed images into a consistent and normalised format, hence improving their suitability for feature extraction and classification tasks. In machine learning models, successful results depend on this preparation since it directly influences the accuracy and efficiency of the subsequent analysis.

Feature Extraction Using Convolutional Layers

Feature extraction using convolutional layers forms a fundamental process in CNNs for visual analysis and interpretation. It is designed to automatically recognise and learn spatial hierarchies of information from the input image—including edges, textures, and patterns—convolutional layers. For duties like image categorisation and object recognition, this capacity makes them rather valuable.

A convolutional layer operates mostly in which a kernel or filter slides across an image to create feature maps. Regarding mathematics, one may define the convolution process as:

$$(I*K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m,j+n) \cdot K(m,n)$$

where:

I - input image,

K - convolutional kernel,

(i,j) - position of the kernel in the image,

M and N - dimensions of the kernel.

This process generates a feature map stressing specifically found objects by the kernel. An image's edges will be highlighted in a feature map by an edge-detecting kernel, for example.

Activation functions then are used to give non-linearity to the output of the convolution procedure. Usually used, the Rectified Linear Unit (ReLU) function has the definition:

$$ReLU(x)=max(0,x)$$

The research define a 2x2 max pooling operation:

$$P(i,j)=\max_{0 \le m<2, 0 \le n<2} F(i \cdot 2+m, j \cdot 2+n)$$

where:

P(i,j) - output of the pooling operation,

F - feature map from the previous convolutional layer.

By means of convolution, activation, and pooling processes, convolutional layers extract hierarchical information from an input image. As more layers are added, they first record low-level information like edges and textures then they catch even more complex patterns and structures. Rich and detailed representations of the image, which the network learns from this hierarchical feature extraction, enable

accurate classification and analysis in applications including osteoarthritis diagnosis from knee X-ray images.

Pseudocode for Feature Extraction Using Convolutional Layers

```
#Step 1: Define Convolutional Layer
def convolutional_layer(input_image, kernel, stride=1, padding=0):
    # Initialize output feature map
    feature_map = zeros(output_height, output_width, num_filters)

    # Convolve each filter over the image
    for filter_index in range(num_filters):
            feature_map[y, x, filter_index] = sum(region * kernel[:, :, filter_index])
   return feature_map

# Step 2: Define Activation Function (ReLU)
def relu_activation(feature_map):
    return max(0, feature_map)

# Step 3: Define Pooling Layer (Max Pooling)
def max_pooling(feature_map, pool_size=2, stride=2):
   pooled_map = zeros(output_height, output_width, num_filters)

    # Apply max pooling
    for filter_index in range(num_filters):
            pooled_map[y, x, filter_index] = max(region)
   return pooled_map

# Step 4: Process Input Image Through Convolutional Layers
def process_image(input_image, kernels):
    # Apply multiple convolutional layers
    conv1 = convolutional_layer(input_image, kernels[0])
    relu1 = relu_activation(conv1)
    pool1 = max_pooling(relu1)

    conv2 = convolutional_layer(pool1, kernels[1])
    relu2 = relu_activation(conv2)
    pool2 = max_pooling(relu2)

    # Add more layers as needed

    return pool2

# Example Usage
input_image = load_image('path_to_image')   # Load image (224x224xchannels)
kernels = [initialize_kernels(filter_count) for _ in range(num_layers)]  # Initialize kernels

feature_map = process_image(input_image, kernels)
```

Feature Sequencing with Recurrent Layers

For RCNNs, feature sequencing with recurrent layers is supposed to capture and explain sequential dependencies inside feature data taken from images. This approach is especially useful when appropriate classification depends on temporal or contextual linkages between features—as in medical imaging where spatial patterns may alter throughout many parts of an image. The research is presented as a sequence of feature vectors, once convolutional layers retrieve features from the images. Among other recurrent layers, Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) cells handle these sequences to find dependencies and patterns stretching across the feature vectors. A 3D tensor (height x width x channels) for the convolutional layer output in an RCNN is first transformed to a 2D matrix whereby each row indicates a feature vector matching a spatial point in the image. Later on, a recurrent neural network (RNN) layer consumes this sequence of feature vectors. One can define the mathematical form of an LSTM cell as follows:

Handling every feature vector in sequence, the LSTM maintains an internal state that catches long-range correlations and temporal trends. This ability to recreate sequential relationships is quite crucial for positions where the interpretation of current features depends on the context provided by past features.

Feature sequencing enhances the comprehension and application of complex data relationships by means of recurrent layers, therefore providing a richer representation for categorisation operations. Applications like medical imaging, where spatial features from numerous sections of the image may be related and demand contextual knowledge for effective diagnosis, find especially relevance for this method.

Pseudocode for Feature Sequencing with Recurrent Layers

```
# Step 1: Define LSTM Cell
def lstm_cell(x_t, h_prev, C_prev, W, b):
    # LSTM parameters
    W_f, W_i, W_C, W_o = W
    b_f, b_i, b_C, b_o = b

    # Forget Gate
    f_t = sigmoid(np.dot(W_f, np.concatenate([h_prev, x_t])) + b_f)

    # Input Gate
    i_t = sigmoid(np.dot(W_i, np.concatenate([h_prev, x_t])) + b_i)
    C_tilde = tanh(np.dot(W_C, np.concatenate([h_prev, x_t])) + b_C)

    # Update Cell State
    C_t = f_t * C_prev + i_t * C_tilde

    # Output Gate
    o_t = sigmoid(np.dot(W_o, np.concatenate([h_prev, x_t])) + b_o)
    h_t = o_t * tanh(C_t)

    return h_t, C_t

# Step 2: Define the LSTM Layer
def lstm_layer(feature_sequences, W, b):
    # Initialize hidden state and cell state
    h_prev = np.zeros(hidden_units)
    C_prev = np.zeros(hidden_units)

    # Process each feature vector in the sequence
```

```
    for x_t in feature_sequences:
        h_prev, C_prev = lstm_cell(x_t, h_prev, C_prev, W, b)

    return h_prev

# Step 3: Process Feature Map Through LSTM
def process_features_with_lstm(feature_map, lstm_weights, lstm_biases):
    # Flatten the feature map into a sequence of feature vectors
    feature_sequences = flatten_feature_map(feature_map)

    # Pass the sequence through the LSTM layer
    final_hidden_state = lstm_layer(feature_sequences, lstm_weights, lstm_biases)

    return final_hidden_state

# Example Usage
feature_map = extract_features_from_image('path_to_image')  # Output of convolutional
layers
lstm_weights = initialize_lstm_weights(hidden_units)
lstm_biases = initialize_lstm_biases(hidden_units)

final_hidden_state = process_features_with_lstm(feature_map, lstm_weights, lstm_biases)
```

Classification

Classification in neural networks is the process of assigning a label or category depending on the characteristics of an input. This is critically essential for tasks like determining the degree of osteoarthritis from knee X-ray images. Usually following feature extraction and feature sequencing, classification uses the neural network learning to meaningfully represent the input data.

Following convolutional and recurrent layer extraction and sequencing, fully connected (dense) layers bind these features to class labels following. Every neurone in a completely linked layer connects to every neurone in the layer below, therefore enabling complex combinations of characteristics. The output of a fully connected layer can be stated mathematically as:

$$z_j = \sum_i w_{ji}x_i + b_j$$

where:

z_j - output of the j^{th} neuron in the fully connected layer,

w_{ji} - weight connecting the i^{th} neuron in the previous layer to the j^{th} neuron,

x_i - output of the i^{th} neuron in the previous layer,

bj - bias term for the j^{th} neuron.

Activation Function:

The non-linearity introduces by passing the output of the fully linked layer via an activation function. In classification issues, the softmax activation function is very well-known. It converts the logit raw output scores into probability. The softmax of the i^{th} class is defined as:

$$pi = e^{zi} / \sum_j e^{zj}$$

where:

p_i - probability of the input belonging to class i,

zi - raw score (logit) for class i,

The denominator sums over all classes j to normalize the scores.

Loss Function:

To enable the model be trained, categorical cross-entropy loss comes out as follows:

$$\text{Loss} = -\sum_I y_i \log(p_i)$$

where:

- Y_i - true label (one-hot encoded) for class i,
- p_i - predicted probability for class i.

Training lets the weight and bias of the model be adjusted to lower the loss function. This is achieved by computing gradients and changing the optimisation method settings, Adam included. Following training, the model may project the class label for new input data by computing the class probabilities and selecting the class with the highest probability. This is stated as:

$$y' = \arg\max_i pi$$

where:

y' - predicted class label,

$\arg\max_i$ - selects the index of the maximum probability p_i.

Pseudocode for Classification

```
# Step 1: Define Fully Connected Layer
def fully_connected_layer(input_vector, weights, biases):
        output_vector = np.dot(weights, input_vector) + biases
    return output_vector
# Step 2: Define Loss Function (Categorical Cross-Entropy)
   # Calculate the loss for each class
    loss = -np.sum(true_labels * np.log(predicted_probs + 1e-9))  # Adding epsilon for
numerical stability
    return loss

# Step 3: Define Prediction Function
 Apply the softmax activation function to get class probabilities
    probabilities = softmax(logits)

    # Predict the class with the highest probability
    predicted_class = np.argmax(probabilities)

    return predicted_class, probabilities

# Example Usage
# Assume feature_vector is the output from the LSTM or feature extractor
feature_vector = extract_features('path_to_image')  # Output from previous layers

# Initialize fully connected layer weights and biases
fc_weights = initialize_fc_weights(num_classes)
fc_biases = initialize_fc_biases(num_classes)

# Make predictions
predicted_class, predicted_probs = predict(feature_vector, fc_weights, fc_biases)
```

```
# Compute the loss for training
true_labels = one_hot_encode(true_class_label, num_classes)
loss = categorical_cross_entropy_loss(predicted_probs, true_labels)
```

3. Results and discussion

This work implements the RCNN model with high-performance computation using TensorFlow. It is evaluated against present techniques including CNN, DNN, and DCNN. The RCNN model far outperformed CNN, DNN, and DCNN models across all three. Strong though they are, CNNs and DNNs occasionally missed complex data patterns. DCNNs lagged behind in exactly grading OA degree even if they outperformed RCNN in managing temporal dependencies and spatial feature correlations.

Table 1: Experimental Setup

Parameter	Value
Dataset Size	9,786 images
Image Resolution	224 × 224 pixels
Train-Validation-Test Split	70%/15%/15%
Model Architecture	RCNN
Recurrent Layer Type	LSTM
Convolutional Layer Type	Standard 2D Convolution
Number of Epochs	50
Batch Size	32
Optimizer	Adam
Learning Rate	0.001
Activation Function	ReLU
Dropout Rate	0.5
Loss Function	Categorical Cross-Entropy
Performance Metric	Accuracy, Precision, Recall, F1-score
GPU Used	NVIDIA Tesla V100

Dataset

Usually affecting the knee joint, osteoarthritis is a condition defined by articular cartilage deterioration, thereby usually shielding bones from impact and friction. The condition can damage the underlying bone and surrounding soft tissues and present in varying degrees of severity.

Comprising 9,786 X-ray images, the dataset—organized by the Osteoarthritis Initiative (OAI)—is kept on Kaggle and shows consistently 224 × 224 pixels every image in the set. The five degree of severity the images fall into are guided by the KL grading system.

Table 2: Data Split and Distribution

Severity Level	Description	Percentage of Dataset
Grade 0	Healthy knee	~40%
Grade 1	Doubtful joint narrowing with possible osteophytes	~18%
Grade 2	Minimal osteophytes with possible joint narrowing	~26%
Grade 3	Moderate osteoarthritis with mild sclerosis	~13%
Grade 4	Severe osteoarthritis with significant sclerosis	~3%

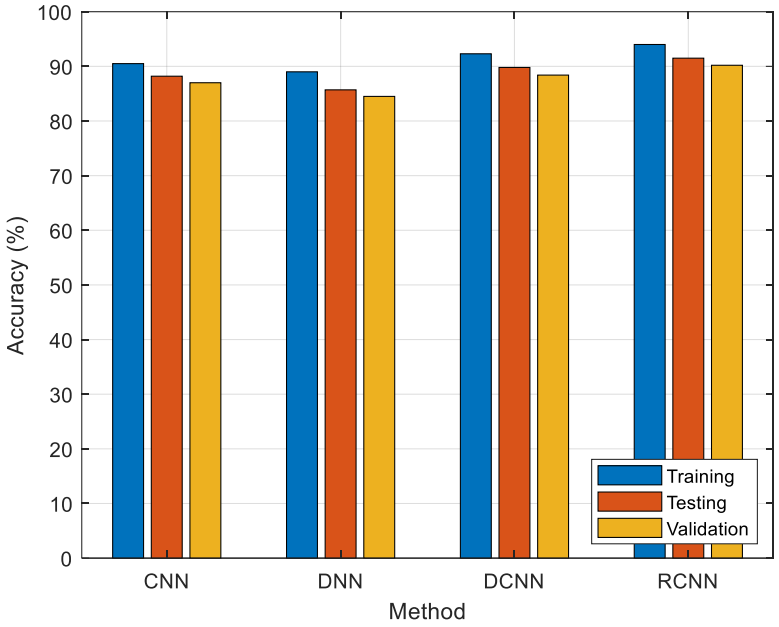

Figure 2: Accuracy (%)

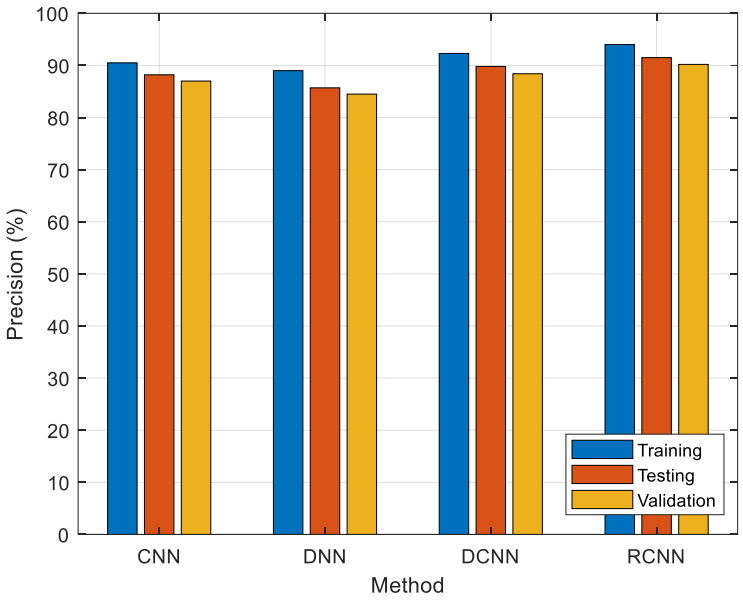

Figure 3: Precision (%)

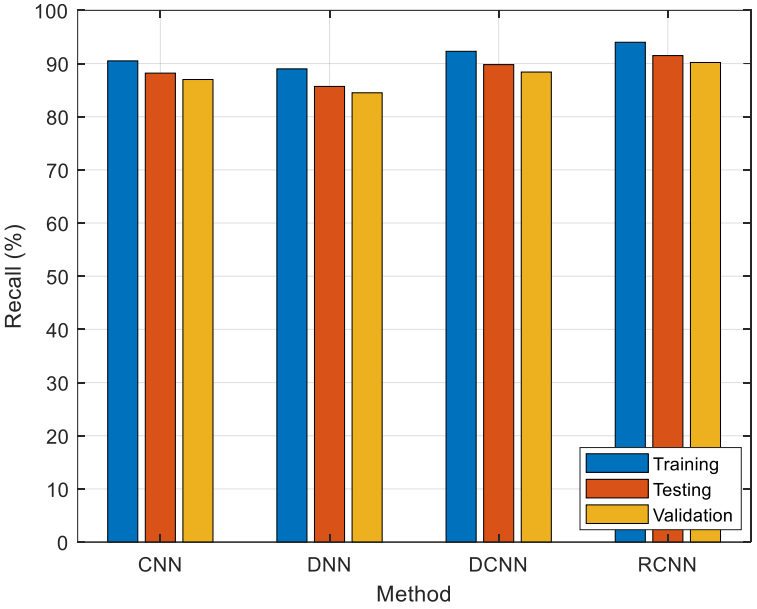

Figure 4: Recall (%)

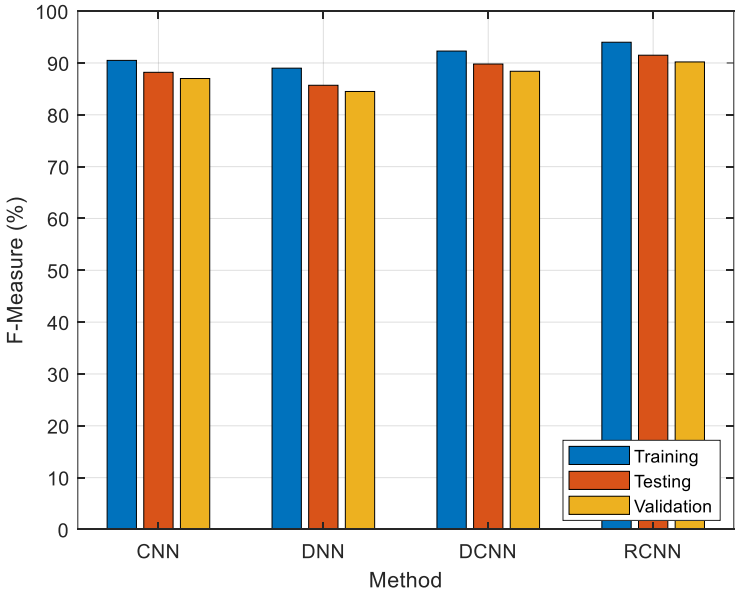

Figure 5: F-Measure (%)

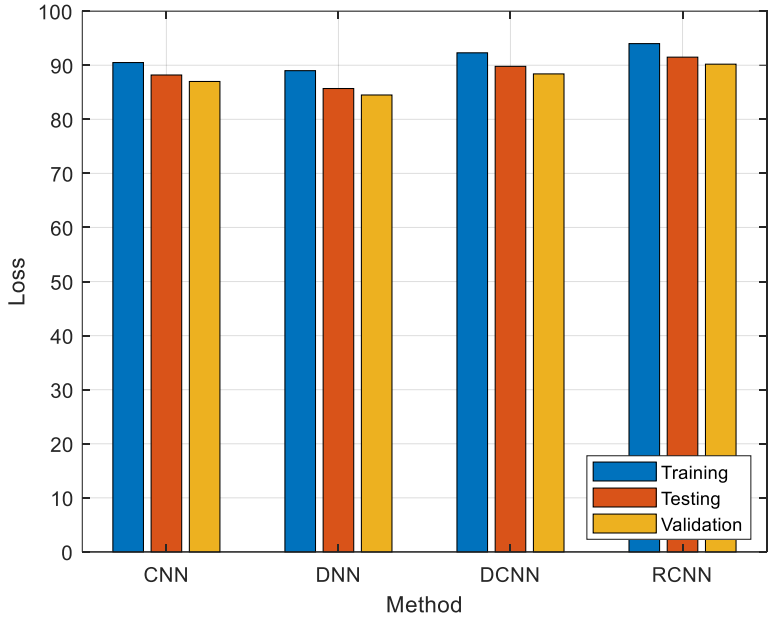

Figure 6: Loss

The existing approaches—CNN, DNN, DCNN, and RCNN—across training, testing, and validation datasets—have their performance compared in figure 2–6 using measures including accuracy, precision, recall, F-measure, and cross-entropy loss.

Accuracy measures the relative correctness of the model for predictions. In the training phase the RCNN technique achieves the best accuracy of 94.0% over DCNN (92.3%), DNN (89.0%), and CNN (90.5%). This greater performance implies that RCNN learns and generalises from the training data more successfully. In testing RCNN maintains its lead over DCNN (89.8%), DNN (85.7%), and CNN (88.2%), with an accuracy of 91.5%. This suggests that RCNN's efficiency transcends unprocessed data to training. Validation accuracy also shows RCNN at 90.2%, following the similar pattern, therefore underlining its durability in many data contexts.

Precision reflects the ability of the model to among all the events it marks as positive precisely identify positive examples. Starting at 93.5% in training and 90.8% in testing, RCNN demonstrates the best precision across all datasets—above DCNN (91.0% and 88.5%), DNN (87.9% and 84.9%), and CNN (89.7% and 87.5%. This implies that RCNN is more constant in lowering false positives. RCNN maintains its edge at 89.4%, hence the validation dataset follows the same pattern. Applications where false positives could have significant consequences depend on high accuracy.

Recall tests if the model can find every relevant positive case. Once more ranking first with a recall of 94.5% during training and 92.0% during testing, Outperforming DCNN (93.5% and 90.5%), DNN (90.1% and 86.4%), and CNN (91.2% and 88.8%). RCNN In the validation phase, RCNN displays a 90.8% recall, therefore verifying its ability to identify genuine positives. In medical diagnostics, for example, good recall is absolutely vital since failing to identify positives could lead to serious errors.

The F-Measure presents a harmonic view of accuracy and recall. With an F-measure of 94.0% in training, 91.4% in testing, and 90.1% in validation RCNN excels. This implies that, in all phases, RCNN provides a good compromise between recall and accuracy. DCNN follows with F-measure of 92.2% in training, 89.5% in testing, and 88.1% in validation. Reduced F-measure values of DNN and CNN imply a compromise between recall and accuracy.

Loss gauges the change in predicted from actual likelihood. Reduced loss values point to better model performance. RCNN scores lowest among the models with 0.22 in training, 0.30 in testing, and 0.32 in validation; this implies that RCNN is not only generating more confident but also more accurate

forecasts. Following with a loss of 0.28 (training), 0.35 (testing), and 0.38 (validation), DCNN comes with Higher loss values of CNN and point to less confident predictions and less accuracy.

Table 3: Performance for RCNN over Training, testing and validation

Dataset	Size (Images)	Accuracy (%)	Precision (%)	Recall (%)	F-Measure (%)	Loss (Cross-Entropy)
Training	600	87.5	85.2	89.1	87.1	0.45
	1200	89.2	87.8	90.3	89.0	0.38
	1800	90.8	88.5	91.2	89.8	0.33
	2400	91.5	89.0	91.8	90.4	0.28
	3000	92.0	89.7	92.1	90.9	0.22
Testing	250	84.0	82.5	85.0	83.7	0.52
	500	86.7	84.5	87.1	85.8	0.45
	750	88.1	85.5	88.3	86.9	0.40
	1000	89.2	86.2	89.1	87.6	0.36
	1250	89.8	87.0	89.8	88.4	0.33
Validation	250	83.5	81.8	84.2	82.9	0.55
	500	85.0	83.2	86.0	84.6	0.50
	750	86.4	84.0	87.2	85.6	0.46
	1000	87.1	84.8	87.8	86.2	0.43
	1250	87.8	85.5	88.4	86.9	0.40

Table 3 shows that the number of training images rises from 600 to 3000, accuracy climbs from 87.5% to 92.0%, reflecting the improved capacity of the model to precisely classify images with extra data. Likewise showing increases in accuracy and memory; accuracy increases from 85.2% to 89.7% and remember from 89.1% to 92.1%. New data training allows the model to reduce false positives and increase actual positive detection ability. Concurrently, the loss ranges from 0.45 to 0.22, suggesting better model performance and a reduced prediction error. Test images grow and the accuracy of the model rises from 84.0% to 89.8% and the precision rises from 82.5% to 87.0%. Recall also increases from 85.0% to 89.8%, suggesting that additional testing data helps to fairly assess the model's performance. From 0.52 to 0.33 the loss drops with more consistent forecasts and lower error rates using larger test datasets. Validation data reveal similar trends. Accuracy increases from 83.5% to 87.8%; precision increases from 81.8% to 85.5%; recall increases from 84.2% to 88.4%. The model generallyizes better to unseen data as the validation set size increases; the loss lowers from 0.55 to 0.40. Thus, the trends over training, testing, and validation datasets show that extending the dataset size enhances the performance of the model, so demonstrating the efficacy of the RCNN method in learning and generalising from larger datasets.

4. Conclusion and future scope

Analysing the performance of the RCNN model over several dataset sizes for training, testing, and validation exposes many significant findings underlining the efficiency of the network. With 600 to 3000 as the count of training images, accuracy increases from 87.5% to 92.0%, precision from 85.2% to 89.7%, and recall climbs from 89.1% to 92.1%. These results show the model's growing ability to learn from more data, hence reducing cross-entropy loss from 0.45 to 0.22. More training examples provide evidence that the RCNN method effectively detects complex patterns and generalises well to new data. Accuracy increases from 84.0% to 89.8%, precision from 82.5% to 87.0%, recall from 85.0% to 89.8%, loss running from 0.52 to 0.33. Since the RCNN model performs more consistently with larger test datasets, these results imply improved generalisation and robustness in predicting unseen data. From 83.5% to 87.8%, precision from 81.8% to 85.5%, recall from 84.2% to 88.4%, and loss lowers from 0.55 to 0.40 in the validation phase as well. This consistency among many datasets guarantees the ability of the model to maintain low error rates and high performance standards.

Reference

[1] Elbaz, A., Mor, A., Segal, G., Debi, R., Shazar, N., & Herman, A. (2014). Novel classification of knee osteoarthritis severity based on spatiotemporal gait analysis. *Osteoarthritis and cartilage*, *22*(3), 457-463.

[2] Yuvaraj, N., Rajput, K., Suganyadevi, K., Aeri, M., Shukla, R. P., & Gurjar, H. (2024, May). Multi-Scale Object Detection and Classification using Machine Learning and Image Processing. In *2024 Second International Conference on Data Science and Information System (ICDSIS)* (pp. 1-6). IEEE.

[3] Ahmed, S. M., & Mstafa, R. J. (2022). Identifying severity grading of knee osteoarthritis from x-ray images using an efficient mixture of deep learning and machine learning models. *Diagnostics*, *12*(12), 2939.

[4] Shesayar, R., Agarwal, A., Taqui, S. N., Natarajan, Y., Rustagi, S., Bharti, S., ... & Sivakumar, S. (2023). Nanoscale molecular reactions in microbiological medicines in modern medical applications. *Green Processing and Synthesis*, *12*(1), 20230055.

[5] Mahendrakar, P., Kumar, D., & Patil, U. (2024). Comprehensive Study on Scoring and Grading Systems for Predicting the Severity of Knee Osteoarthritis. *Current Rheumatology Reviews*, *20*(2), 133-156.

[6] Jain, R. K., Sharma, P. K., Gaj, S., Sur, A., & Ghosh, P. (2024). Knee osteoarthritis severity prediction using an attentive multi-scale deep convolutional neural network. *Multimedia Tools and Applications*, *83*(3), 6925-6942.

[7] Helwan, A., Azar, D., & Abdellatef, H. (2022). An update on the knee osteoarthritis severity grading using wide residual learning. *Journal of X-ray Science and Technology*, *30*(5), 1009-1021.

[8] Vinuja, G., Saravanan, V., Maharajan, K., Jayasudha, V., & Ramya, R. (2024). Diagnostic Device for Sustainable Medical Care Using Hyperspectral Imaging. In *Emerging Advancements in AI and Big Data Technologies in Business and Society* (pp. 128-142). IGI Global.

[9] Jain, R. K., Sharma, P. K., Gaj, S., Sur, A., & Ghosh, P. (2024). Knee osteoarthritis severity prediction using an attentive multi-scale deep convolutional neural network. *Multimedia Tools and Applications*, *83*(3), 6925-6942.

[10] Sakthisudhan, K., Saranraj, N., Vinothini, V. R., Sekaran, R. C., & Saravanan, V. (2024). A Novel CAD Structure with Bakelite Material-Inspired MRI Coils for Current Trends in an IMoT-Based MRI Diagnosis System. *Journal of Electronic Materials*, 1-14.

[11] Antony, J., McGuinness, K., Moran, K., & O'Connor, N. E. (2017). Automatic detection of knee joints and quantification of knee osteoarthritis severity using convolutional neural networks. In *Machine Learning and Data Mining in Pattern Recognition: 13th International Conference, MLDM 2017, New York, NY, USA, July 15-20, 2017, Proceedings 13* (pp. 376-390). Springer International Publishing.

[12] Chen, P., Gao, L., Shi, X., Allen, K., & Yang, L. (2019). Fully automatic knee osteoarthritis severity grading using deep neural networks with a novel ordinal loss. *Computerized Medical Imaging and Graphics*, *75*, 84-92.

[13] Antony, J., McGuinness, K., O'Connor, N. E., & Moran, K. (2016, December). Quantifying radiographic knee osteoarthritis severity using deep convolutional neural networks. In *2016 23rd international conference on pattern recognition (ICPR)* (pp. 1195-1200). IEEE.

[14] Tiulpin, A., & Saarakkala, S. (2020). Automatic grading of individual knee osteoarthritis features in plain radiographs using deep convolutional neural networks. *Diagnostics*, *10*(11), 932.

[15] Tiwari, S. Knee Osteoarthritis Dataset with Severity Grading. *Kaggle. Available online: https://www. kaggle. com/datasets/shashwatwork/knee-osteoarthritis-dataset-with-severity*.