Chapter 6
DRCNN – Depth Restricted Convolution Neural Network

The existing popular models do not give good classification accuracy even after fine-tuning on TGFD. In this chapter, a more effective and efficient model has been proposed. Also, the selection of hyperparameters, an optimizer and a learning rate for the proposed model is discussed. The chapter ends by comparing the results of proposed model with various existing models considering the evaluation parameters accuracy, precision, recall and f1 score.

6.1 INTRODUCTION

All the pre-built deep convolutional neural networks like VGG16, Inceptionv3, VGG19 have large no. of layers and, hence contain a huge number of parameters [140]. Such networks need hours, days, or even weeks to train. To reduce the time necessary to train a model with increased classification accuracy, a lightweight network model with fewer parameters and a small convolutional kernel size is required.

This research work has tried to develop a more accurate and efficient model for TGFD with less number of parameters. Also, the highest classification accuracy for TGFD achieved by Inceptionv3 is 86.22% after transfer learning and 89.36 % after fine-tuning which can be further improved.

To design a new model from scratch, the very first step is to choose the proper hyperparameter for the model. To make the model lightweight, it is necessary to understand what are the hyperparameters and what value should be set for the same. An empirical study have been done for selection of these parameters for the proposed model.

6.2 Hyperparameter Selection Using Empirical Analysis

For any CNN algorithm, it is very necessary to decide the hyperparameters like the number of convolutional layers, the number of fully connected layers, batch size, kernel size, number of filters in a layer, optimizer, learning rate, etc. By doing an exhaustive literature survey, the following observations have been found.

- There are two types of datasets: Deeper and Wider. A deeper dataset has more images per class than a wider one. Deeper architecture works best with deeper datasets and shallow architecture with wider datasets [145]. The proposed dataset, TGFD, is a deeper dataset as it has more number of images per class. Hence, deeper network gives better results with it.
- The number of convolutional and fully connected layers directly affects the runtime of the model [146]. In order to reduce computational complexity less number of convolutional and fully connected layers should be chosen.
- The model results in higher accuracy and lower runtime when convolutional layers have fewer filters [148]. Generally, the number of filters varies between 16,32,64 and 128. A lower number of filters for all the convolutional layers should be chosen for the proposed model.
- The number of filters in convolution layers and the size of filters have a significant effect on the accuracy of the system [148]. The filters size can be typically specified as 3x3 ,5x5,7X7,9X9. The lower filter size should be chosen for making model more accurate.
- The max-pooling layer reduces the parameter count, which decreases computational complexity [149]. Hence, max-pooling layers should be used in order to reduce model complexity of proposed model.
- The accuracy of the model relies more on the quantity of convolution filters within a layer and the dimensions of the convolution kernel, rather than the depth of the network. [148].
- When the learning rate is low, a lower batch size gives a better result [147]. A lower batch size should be chosen for the proposed model as it helps to improve accuracy.

- The Rectified Linear Unit (ReLU) activation function is commonly used in convolutional layers of neural networks for several reasons. It offers advantages over other activation functions such as sigmoid or tanh. ReLU is preferred over other activation functions in convolutional layers due to the following reasons:

  - **Efficiency:** ReLU is computationally efficient compared to other activation functions like sigmoid or tanh. The ReLU function simply sets negative values to zero while leaving positive values unchanged. This simple operation can be implemented efficiently using basic mathematical operations.
  - **Avoiding the vanishing gradient problem:** ReLU helps alleviate the vanishing gradient problem, which is a challenge in deep neural networks. The vanishing gradient problem occurs when the gradients during backpropagation become extremely small, leading to slow convergence and difficulty in training the network. ReLU's derivative is either 0 or 1, ensuring that the gradients are not diminished through repeated multiplications.
  - **Sparse activation:** ReLU introduces sparsity in the network. When the input to a ReLU unit is negative, the output is zero, effectively deactivating the neuron. This sparsity property helps the network focus on the most relevant features and reduces the overall computational burden.
  - **Non-linearity:** ReLU introduces non-linearity, allowing the network to learn complex relationships between inputs and features. Convolutional layers without non-linear activation functions would only perform linear transformations, limiting the network's expressive power. ReLU enables the network to capture intricate patterns and improve its representation capabilities.

- By utilizing ReLU in convolutional layers, neural networks become more efficient, better at handling deep architectures, and capable of learning complex features.
- It is very important to choose the correct optimizer while compiling the model. The optimizer ties together the model parameters like weights, learning rates and loss functions.

- Some of the well-known optimizers are Gradient Descent, Stochastic Gradient Descent (SGD), Adam, RMSprop, Momentum and Adagrad [153][155]. The comparison of some of the most popular optimizers has been shown in Table 6.1.

Table 6.1 Comparison of Optimizers

Optimizer	Description	Advantages	Disadvantages
Adam	Combines advantages of RMSprop and momentum. Uses moving averages of previous gradients and squared gradients to adapt learning rates per parameter.	Suited to a variety of deep learning tasks Rapid convergence Functions well with sparse gradients Strong hyperparameter options.	Needs more memory because there are more moving average parameters Choice of learning rate sensitive
Adagrad	Scaling learning rates per parameter inversely proportional to the sum of past squared gradients allows for parameter adaptation.	Effective with sparse data and gradients Automatic rate adjustments for learning Less susceptible to early learning rate selection	Reduces learning rates too aggressively, making convergence Accumulates the total of historical gradients possibly causing memory problems
RMSprop	Divides the learning rates for each parameter by the exponential moving average of squared gradients to adapt learning rates.	Solved Adagrad's aggressive learning rate decay Faster convergence than standard SGD Suitable for non-stationary goals	Learning rate must be manually adjusted Accumulates past squared gradients potentially causing memory difficulties
SGD	Model parameters are updated by a simple optimization approach based on the gradient of the loss function calculated on a mini-batch.	Simpleness and a minimal memory requirement Simple to implement Can escape from poor local minima	Simpleness and a minimal memory requirement Simple to implement

From the comparison shown in Table 6.1, it has been observed that adam is beneficial as it is a combination of AdaGrad and RMSProp. It requires less memory, computationally efficient and is easy to implement.

- The proposed model is compiled using different optimizers in order to decide which optimizer is best suited for the TGFD. This research has considered Adam, Adagrad, RMSprop and SGD optimizer [153] and the results are shown in Table 6.2.

Table 6.2 Classification accuracy of DRCNN using different optimizers

Optimizer Name	**Accuracy**
Adam	95.48
Adagrad	85.77
RMSprop	76.11
SGD	80.29

From Table 6.2, it can be seen that practically also the 'Adam' optimizer has given the best classification accuracy. Hence, for the proposed model Adam optimizer is used.

- According to Leslie N. Smith, there is a unique term known as "Cyclic Learning Rate" in which one needs to fix the minimum and maximum values for the learning rate and run the model for several epochs with the learning rate varying between the minimum and maximum values [144]. This range test is also known as the learning rate (LR) range test. The minimum and maximum values for the learning rate have been set to 0.1 and 0.3, respectively and the learning rates vary between them [155]. As shown in Table 6.1, after many experiments on different learning rates, the learning rate is set to 0.0001 for the Adam optimizer on TGFD as it gives the highest accuracy. The categorical cross-entropy loss function has been used for error calculation.

Table 6.3 Adam optimizer with different learning rates for DRCNN

Learning Rate	Accuracy	Loss
0.1	91.65	5.25
0.2	85.36	4.56
0.3	76	4.12
0.01	80.34	3.89
0.02	82.35	4.79
0.03	81.78	3.45
0.001	93.45	2.21
0.002	91.66	2.78
0.003	81.92	2.56
0.0001	**95.48**	**0.8**
0.0002	89.6	1.45
0.0003	85.56	1.58
0.00001	70.65	2.36
0.00002	65.58	0.58
0.00003	62.45	1.87

- KerasTuner is a tool that helps to configure model hyperparameters [150-151]. KerasTuner has built-in algorithms for random search algorithms, Bayesian Optimization and Hyperband [152]. It is a useful package of Keras that does the experiments quickly and helps in tuning the hyperparameters such as the number of convolutional layers, input neurons in each convolutional layer and filter size. An experiment has been implemented using KerasTuner to decide the approximate range of a number of convolutional and pooling layers as shown in the code below.

```
def build_model(hp):
    model = keras.Sequential()
    for i in range(hp.Int('num_layers', 2, 20)):
        model.add(layers.Dense(units=hp.Int('units_' + str(i),
                                            min_value=32,
                                            max_value=512,
                                            step=32),
                               activation='relu'))
    model.add(layers.Dense(5, activation='linear'))
model.compile(optimizer=keras.optimizers.Adam()),
            loss='categorical_crossentropy',
            metrics=['accuracy'])
  return model
```

```
tuner = RandomSearch(
    build_model,
    objective='val_mean_absolute_error',
    max_trials=5,
    executions_per_trial=3,
    directory='project',
    project_name='DRCNN')
```

6.3 Proposed DRCNN Model

Based on the above observations a model from scratch has been proposed. To make the model lightweight it has been constructed with 11 Convolutional layers, 4 Max-pooling layers followed by one fully connected layer and a SoftMax layer. It is a deep architecture and since it restricts the depth by having less number of convolutional and fully connected layers, it named as "The Depth restricted Convolutional neural network (DRCNN)". Table 6.4 shows the parameters selected for the proposed model which helps to increase an accuracy and reduce the computational complexity.

Table 6.4 Hyperparameters for DRCNN

Parameters	Chosen Value for DRCNN
Convolutional layers	11
Max pooling layers	4
Fully Connected layers	1
Batch size	32
No. of filters in convolutional layers	16,32,64,128
Filter size	3X3, 5X5
Learning rate	0.0001

The architecture of the proposed model DRCNN is shown in Fig. 6.1.

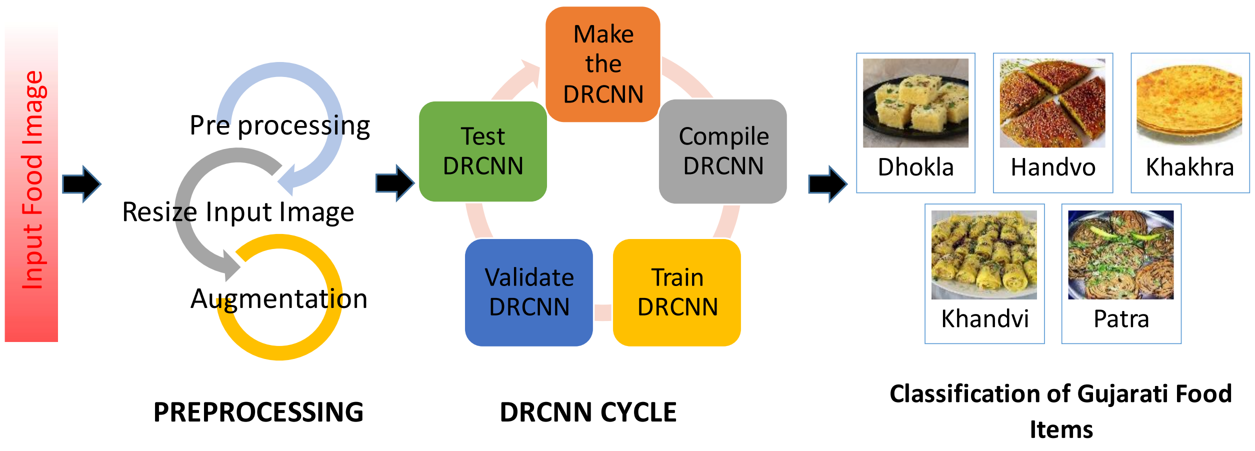

Fig. 6.1 Architecture of the proposed model DRCNN

As seen in Fig. 6.1, DRCNN consists of following steps:

1. Dataset Creation:

The Gujarati food images collected in TGFD, has been given as input images to the model.

2. Preprocessing:

The images are first preprocessed by the following steps.

- The images are resized to 224x224 before processing.
- Data augmentation techniques have been applied to artificially increase the size of the dataset. The total number of images after applying data augmentation techniques is 37,044 in TGFD. The dataset is divided into training, validation and testing with 70%, 20%, and 10% respectively using the Python library Splitfolders.
- The images are preprocessed using the proposed ISMF algorithm to remove noise from images. The denoising images given to the model.

3. Build the model:

The model is built with the hyperparameter selected.

- The proposed model first extracts a data feature by 5 x 5 convolutional with 16 filters which are followed by one more Convolutional layer of 5 x 5 Convolutional and 32 filters, followed by a pair of batch normalization (BN) and Relu. A 2x2 max-pooling layer with stride 2 has been applied after each BN in the model.
- The next 3 Convolutional layers have been added of Convolutional 3x3, out of which the first layer uses 16 filters and the rest two contain 32 filters. After these, 3 Convolutional layers have been added of Convolutional 3x3, out of which the first layer uses 32 filters and the rest two contain 64 filters.
- A more 3 Convolutional layers have been added of convolutional 3x3, out of which the first layer uses 64 filters and the rest two contain 128 filters. A flatten layer has been added after the last max-pooling layer to make multidimensional output linear and to pass it onto a fully connected layer. Lastly, one fully connected layer has been added followed by a Softmax output layer. When there are more than two classes Softmax is used as it returns probabilities of each class. The total trainable parameter in the model is 481,557. The detailed structure of DRCNN has been shown in Fig.6.2.

Model: Depth Restricted Convolutional Neural Network		
Layer	**Output Shape**	**Param #**
Conv 1	(None,224,224,16)	1216
Conv 2	(None,224,224,32)	12832
batch normalization	(None,224,224,32)	128
Max pool 1	(None,112,112,32)	0
Conv 3	(None,112,112,16)	4624
Conv 4	(None,112,112,32)	4640
Conv 5	(None,112,112,32)	9248
batch normalization	(None,112,112,32)	128
Max pool 2	(None,56,56,32)	0
Conv 6	(None,56,56,32)	9248
Conv 7	(None,56,56,64)	18496
Conv 8	(None,56,56,64)	36928
batch normalization	(None,56,56,64)	256
Max pool 3	(None,28,28,64)	0
Conv 9	(None,28,28,64)	36928
Conv 10	(None,28,28,128)	73856
Conv 11	(None,28,28,128)	147584
batch normalization	(None,28,28,128)	512

Model: Depth Restricted Convolutional Neural Network		
Layer	**Output Shape**	**Param #**
Max pool 4	(None,14,14,128)	0
Flatten	(None,25088)	0
Dropout	(None,25088)	0
Dense Layer	(None,5)	125445
Total params: 482,069 **Trainable params: 481,557** **Non-trainable params: 512**		

Fig. 6.2 Structure of the Depth Restricted Convolutional Neural Network

- The model is trained using the training dataset.
- In the next step the model is validated using the validation dataset to monitor the model's performance on unseen data.

4. Compile the model:

Once the training and validation steps are completed the model is compiled for several epochs using the Adam optimizer with a learning rate of 0.0001.

5. Test the model:

The final step is to test the model on the test dataset. It gives an unbiased assessment of model's accuracy on unseen data. Model evaluation parameters like accuracy, F1 Score, precision and recall are used to evaluate the model [150,154].

- **Precision:** Measures the accuracy of the model's positive predictions [154]. It emphasizes the significance of the positive predictions and illustrates how many of the projected positive cases are true positives. In simple terms it defines the actual positive results out of the total positive results predicted by the model. The precision can be calculated using below formula:

  Precision = True Positives / (True Positives + False Positives) (6.1)

  High accuracy shows that the model has a low rate of false positives, implying that it properly identifies positive occurrences while minimizing the number of examples labelled as positive mistakenly.

- **Recall:** Also known as sensitivity, defines the classifier's completeness [154]. In other words, recall measures how well a model can "recall" positive examples from the entire collection of positive examples in the dataset. The recall can be calculated using below formula:

  Recall = True Positives / (True Positives + False Negatives) (6.2)

  A high recall value suggests that the model is properly classifying the majority of positive cases and effectively identifying a large proportion of positive instances.

- **F1 Score:** The F1 score combines both precision and recall into a single value [154]. The F1 score can be calculated using below formula:

  F1 score = 2 * (Precision*Recall) / (Precision + Recall) (6.3)

  The F1 score ranges from 0 to 1, with a higher value indicating better performance. When the dataset is unbalanced or when recall and precision are equally crucial, it is a helpful metric. The F1 score recognizes models with a good combination of precision and recall.

The algorithm of DRCNN has been shown in Fig.6.3

<u>Algorithm Steps:</u>

Input:

Training, Validation and Testing instance set 'T', an Image set and a label value

Image Set I(i) = {I1(i), I2(i),…..,In(i)}

Label Set L(i) = {Class1, Class2, ------, Class n}

Initialization:

Step 1: Collect the images from mobile, Internet and Real images clicked by visiting restaurants to prepare the dataset.

Pre-processing Phase:

Step 2: For each instance of input data,

```
        Remove Background Noise
        Resize the image into the specified range
```

Step 3: Apply augmentation techniques to increase the size of data artificially

Define the model:

Step 4: Define the model Depth Restricted convolutional neural network

Step 5: Extract the input, output and intermediate properties of layers in the model

Step 6: *Configure the model and* load the data in the model DRCNN

Feature Extraction Phase:

Step 7: Apply activation functions to get features dataset from selected or configure layer of the model

Step 8: Prepare feature dataset and its equivalent target label for hyper-parameter tuning, training, and testing of model

Parameter Hyper tuning Phase:

Step 9: Select the optimizer and learning rate for the model

Step 10: Compile the model

Training Phase:

Step 11: Initialize the parameter tuned for a model of DRCNN

Step 12: Initialize the feature data and label data for the training dataset.

Step 13: Train the model for DRCNN algorithms.

Validation Phase:

Step 14: Initialize the feature data for the validation dataset.

Step 15: Validate the model DRCNN

Testing Phase:

Step 16: Initialize the feature data for the testing dataset.

Step 17: Load the trained model of DRCNN algorithms.

Step 18: Check the Testing accuracy of the model to check the model's efficiency.

Fig. 6.3 Algorithm of DRCNN

6.4 Features of DRCNN

DRCNN, a unique architecture, incorporates distinctive features that contribute to its effectiveness in image classification tasks:

- **Recursive Convolutional Layers:** DRCNN utilizes a recursive approach by incorporating recursive convolutional layers. These specialized layers enable the network to iteratively learn and refine features from the input data. By recursively applying convolutional operations, DRCNN can capture hierarchical information and extract complex patterns more comprehensively.
- **Deep Architecture:** DRCNN adopts a deep architecture, featuring multiple layers. The depth of the network allows for the learning of hierarchical representations at various levels of abstraction. This depth facilitates the extraction of intricate and detailed features from the input images.
- **Convolutional Operations:** Central to DRCNN are the convolutional operations. These operations involve the application of filters or kernels to the input data, facilitating local feature extraction. By convolving these filters across the input, DRCNN can effectively capture spatial dependencies and extract meaningful features.
- **Non-linear Activation Functions:** DRCNN employs non-linear activation functions, such as ReLU, which introduce non-linearity into the network. This non-linearity enables DRCNN to learn complex relationships between the input data and their corresponding labels, enhancing its capability to model intricate patterns.
- **Recursive Learning and Refinement:** A key feature of DRCNN is its recursive learning and refinement mechanism. Through recursive convolutional layers, the network iteratively refines its feature representations. This iterative process enables DRCNN to capture increasingly detailed and fine-grained features from the input data.

These unique features collectively contribute to the effectiveness of DRCNN in image classification. The recursive convolutional layers, deep architecture, convolutional operations, non-linear activation functions, skip connections, and recursive learning and refinement enable DRCNN to extract intricate features and model complex patterns, leading to improved classification accuracy.

6.5 Experiments and Results

To see the performance analysis of DRCNN on TGFD, the DRCNN model has been implemented on an Intel i7-9750H Lenovo Legion Y540 CPU @ 2.60GHz processor, which supports a multicore processor equipped with a GeForce GTX 1650 NVIDIA GPU with 8GB of memory. Python 3.8.8 was used in the Deep Learning Framework with Keras 2.7 and TensorFlow 2.7.

DRCNN has been tested with preprocessed images and without preprocessed images on TGFD. The comparison has been done using the accuracy of the model before applying pre-processing techniques and after preprocessed using ISMF.DRCNN runs starting from 10 epochs to 500 epochs. It has been observed that after 100 epochs the performance of the model is not improving, hence for comparison accuracy of 100 epochs has been considered. The accuracy obtained on TGFD by considering images with and without preprocessing has been shown in Table 6.5.

Table 6.5 Accuracy of DRCNN on TGFD

DRCNN	Epoch (Accuracy (%))								
	20	40	60	80	100	200	300	400	500
Before ISMF	56.25	60.02	61.25	65.58	68.89	67.23	67.55	68.25	67.58
After ISMF	86.78	87.30	89.30	91.38	95.48	95.12	95.07	94.23	95.47

Fig. 6.4(a) and Fig. 6.4(b) show the graphical representation of accuracy obtained on TGFD by considering images with and without preprocessing.

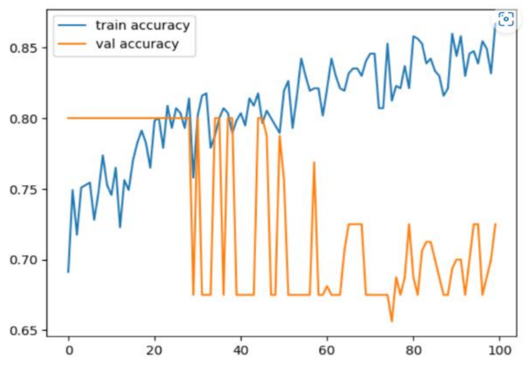

Fig.6.4 (a): Training vs. Validation accuracy before pre-processing

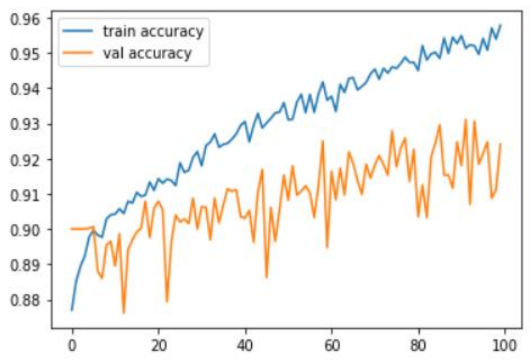

Fig.6.4 (b): Training vs. Validation accuracy after pre-processing using ISMF

From the result shown in the Table 6.5, it has been observed that applying ISMF on the images before processing results in good training accuracy of 95.48% for the model. Before pre-processing the images, the proposed model achieves only an accuracy of 68.89%.

The results achieved by the experiments conducted in sections 4 and 5 in chapter 5 shows that by implementing transfer learning and fine-tuning, the testing accuracy has been increased by at least 5% and 8%, respectively, proving that transfer learning along with fine-tuning significantly improves classification accuracy.

It has been observed from Table 6.5 that after 100 epochs, accuracy is not improving. Hence for comparison accuracy of 100 epochs has been considered. Table 6.6 shows the results for the DRCNN and fine-tuned models run for 100 epochs with classification accuracy along with the number of parameters, time required to train a model and number of convolutional layers and fully connected layers used.

Table 6.6 Comparison of the DRCNN with Fine-tune existing Models

Model Name	Classification Accuracy (%)	No. of Parameters	Conv/FC	Time (In seconds)
VGG16	85.23	13,83,57,544	13-Mar	4000
VGG19	87.3	14,36,67,240	16-Mar	4300
ResNet50	62.32	2,56,36,712	>17/1	14000
Inceptionv3	89.36	2,38,51,784	>60/3	22500
Alexnet	68.73	6,23,78,344	05-Mar	5500
DRCNN	**95.48**	**4,82,069**	**11-Jan**	**3700**

It can be seen from the Table 6.6 that the DRCNN model achieves a remarkable classification accuracy of 95.48%, which is more than all the fine-tuned models. It is more than the highest classification accuracy 89.36% which is achieved by inceptionv3 after fine-tuning. The DRCNN model contains 482069 parameters, which is 48 times less than the Inceptionv3 model. As a result, the DRCNN model takes only 30 minutes to run on the NVIDIA GPU GeForce GTX 1650, which is very low as compared to other pre-built models, which take 50 to 60 minutes to run. The validation loss of DRCNN is only 0.8041.

The graphical representation of accuracy, number of parameters and time taken by each model to run for all the fine-tuned models with the DRCNN is shown in Fig. 6.5,6.6 and 6.7 respectively.

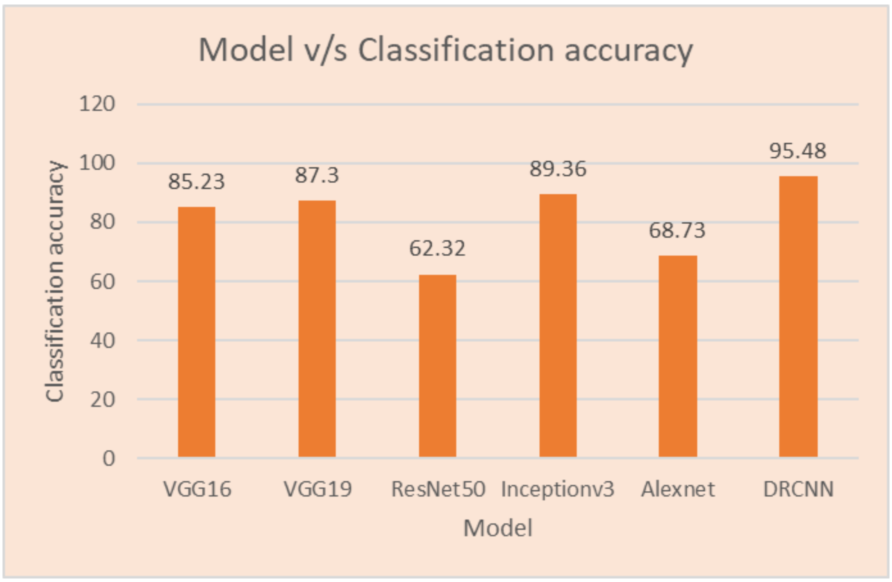

Fig 6.5 Comparison of Fine-tune Models accuracy with the DRCNN

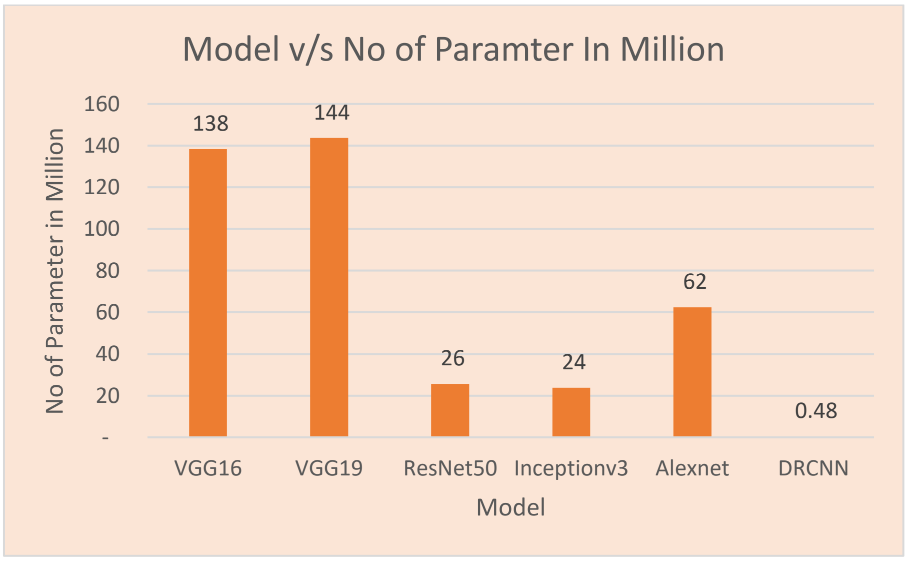

Fig 6.6 Comparison of Fine-tune Models parameters with the DRCNN

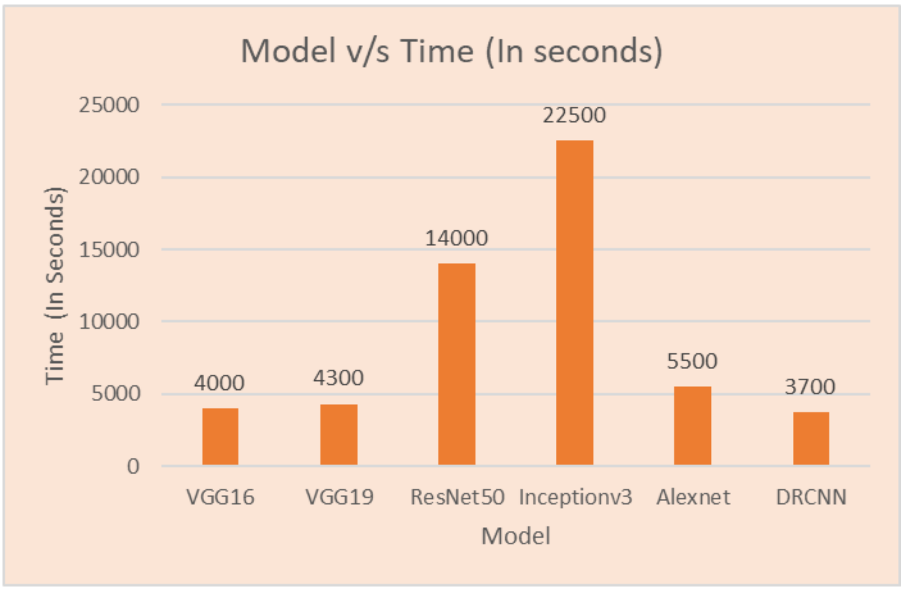

Fig 6.7 Time Comparison of Fine-tune Models with the DRCNN

Table 6.7 shows the performance evaluation metrics for all five prebuilt models with DRCNN.

Table 6.7 Performance of evaluation metrics for prebuilt models and DRCNN

Model	Accuracy (%)	Precision	Recall	F1 Score
VGG16	85.23	0.82	0.93	0.87
VGG19	87.3	0.75	0.86	0.8
Alexnet	68.73	0.81	0.76	0.78
GoogleNet	89.36	0.82	0.88	0.83
Resnet50	62.32	0.92	0.67	0.77
DRCNN	95.48	0.95	0.9	0.92

From Table 6.7 the following observation can be made:

- VGG16 achieves an accuracy of 85.23% and demonstrates a balanced precision of 0.82 and recall of 0.93. This indicates its ability to correctly identify positive samples while minimizing false positives, resulting in an overall F1 score of 0.87.
- VGG19 performs slightly better than VGG16, with an accuracy of 87.3%. However, it shows a lower precision of 0.75 and an F1 score of 0.80, indicating a relatively

higher rate of false positives. The recall of 0.86 suggests its effectiveness in capturing a high proportion of positive samples.

- Alexnet achieves an accuracy of 68.73%, the lowest among the listed models. It exhibits a relatively high precision of 0.81 but a lower recall of 0.76, indicating a higher rate of false negatives. The F1 score of 0.78 reflects its moderate overall performance.
- Inceptionv3 stands out with the highest accuracy of 89.36%. It demonstrates balanced precision (0.82) and recall (0.88), indicating its ability to correctly identify positive samples and avoid false negatives. The overall F1 score of 0.83 reflects its high performance.
- Resnet50 exhibits the lowest accuracy of 62.32% among the listed models. It shows high precision (0.92) but a lower recall (0.67), indicating a relatively higher rate of false negatives. The F1 score of 0.77 suggests a moderate overall performance.
- DRCNN showcases the highest accuracy of 95.48%. It demonstrates both high precision (0.95) and recall (0.90), indicating its proficiency in correctly identifying positive samples while minimizing false positives and negatives. The F1 score of 0.92 signifies its excellent overall performance.

In conclusion, the observations reveal that DRCNN exhibit the best performance among the listed models.

The training and testing accuracy for DRCNN is shown in Fig. 6.8 for 100 epochs.

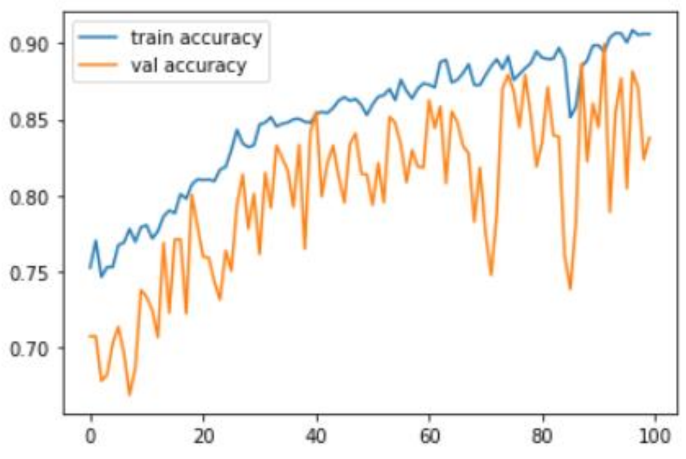

Fig. 6.8 Accuracy curves for DRCNN with Training epoch set to 100

6.6 Testing DRCNN

The DRCNN is tested in two ways to check its effectiveness and performance. In the first test case, the DRCNN is run for a higher number of Gujarati food classes. The DRCNN is run for different food datasets in the second test case to see its effect on other types of datasets.

6.6.1 Performance Evaluation of DRCNN on Extended TGFD

Initially, TGFD consists of five food items. Later the TGFD has been extended to include more Gujarati food items from 5 food items to 7,10,12,15 and up to 20 food items. Extended TGFD food classes with a number of images in the dataset are as shown in Table 6.8:

Table 6.8 Number of images per Food Class in Extended TGFD

Sr. No.	Food Class	Number of images
1	Muthiya	99
2	Khichu	100
3	Poha	110
4	Thepla	104
5	Chapati	400
6	Puri	400
7	White Rice	400
8	Idly	400
9	Dabeli	108
10	Biryani	400

Sr. No.	Food Class	Number of images
11	Gulab Jamun	170
12	Samosa	400
13	Salad	400
14	Upma	400
15	Gujarati Dal	400
16	Dhokla	377
17	Handvo	367
18	Khakhra	295
19	Khandvi	419
20	Patra	306

The total number of images in Extended TGFD are 6055. Data augmentation techniques have been applied to TGFD to artificially increase the size of the dataset. The total number of images in extended TGFD after applying data augmentation techniques is 1,27,155. The dataset is then divided into training, validation and testing with 70%, 20%, and 10% respectively using the Python library Splitfolders.

The images from extended TGFD are preprocessed using the proposed ISMF algorithm to remove noise from images. The denoising images are given as an input to the model.

All the prebuilt models and DRCNN are tested on Extended TGFD. The DRCNN is run in the same environment, compiled with Adam optimizer at a learning rate of 0.0001 for starting from 10 epochs to 500 epochs, but it has been observed that after 100 epochs, accuracy is not improving. Hence, for comparison, the accuracy for 100 epochs has been considered in Table 6.9. The same experiments were conducted on all fine-tuned models to compare the results. The results obtained using prebuilt models and DRCNN on extended TGFD are shown in Table 6.9.

Table 6.9 Comparison of Fine-tuned Prebuilt models and DRCNN for extended TGFD

No. of Gujarati food class	No. of Gujarati food images after Augmentation	VGG16 (%)	VGG19 (%)	Inceptionv3 (%)	Alexnet (%)	Resnet50 (%)	DRCNN (%)
5	36687	85.23	87.3	**89.36**	68.73	62.32	**95.48**
7	42336	76.21	73.79	81.65	70.23	58.54	**91.12**
10	48531	70.59	73.95	75.76	71.24	55.54	**93.36**
12	65310	70.44	72.01	77.61	72.45	54.85	**92.98**
15	90510	72.37	66.64	85.7	65.54	52.21	**93.76**
20	127155	75.49	70.23	67.5	69.32	50.25	**96.10**

From the results seen in Table 6.9, the following observations are made:

- VGG16 and VGG19 show consistent performance across different number of classes of food, however their accuracy is not satisfactory.
- Alexnet, ResNet50 and Inceptionv3 show variable performance with increasing number of classes. The model may struggle with more complex classification.
- DRCNN outperforms the fine-tuned Inceptionv3 model for more number of food classes in terms of accuracy.
- The accuracy of DRCNN has been increased as the number of food classes are increased shows model's capabilities to handle complex tasks.
- DRCNN shows superior performance, even with a smaller number of classes, indicating its effectiveness in the Gujarati food classification task.

6.6.2 The Performance of DRCNN with Different Food Datasets

Five different existing datasets having different types of food items have been considered. DRCNN along with all five fine tune models runs for selected five datasets namely Food20, Indian-100, Food-101, FFML and UECFOOD-100 to check DRCNN's versatility. Details of chosen dataset has already been described in section 2.4.

Table 6.10 shows the result of DRCNN and prebuilt models for different datasets.

Table 6.10 Result of DRCNN for Different Datasets

Name of Dataset	Type of Food item in the dataset	No. of food class	No. of food images	Accuracy of VGG16 (%)	Accuracy of VGG19 (%)	Accuracy of Inceptionv3 (%)	Accuracy of Alexnet (%)	Accuracy of Resnet50 (%)	Accuracy of DRCNN (%)
Food20[2]	Indian	20	2000	56.62	70.78	70.56	19.32	45.58	**95.5**
Indian-100[3]	Indian	50	5000	57.23	58.12	65.47	27.4	17.12	**97.7**
Food-101 [157]	All Mix types of Food	101	101000	67.34	69.34	53.38	35.25	35.89	**98.98**
FFML Dataset [158]	Romanian food dishes	424	1281	47.12	54.23	48.01	23.45	17.26	**99.79**
UECFOOD100 [159][174]	Japanese food	100	14461	54.67	57.89	60.21	50.12	57.12	**99.10**

From the results, it has been observed that:

- Existing pre-built models do not give good results on all datasets, but DRCNN gives remarkable accuracy on all types of datasets, especially on FFML and UECFOOD10.
- Alexnet gives poor performance on all types of datasets because it is a shallow architecture.
- Resnet50 model begins to overfit after 60 epochs as it as it has been trained on very large-scale dataset
- VGG16 and VGG19 give good performance for FOOD20 but for the rest of the dataset, the performance is not good.
- Inceptionv3 fails to give good accuracy in almost all other types of datasets.

It can be concluded from the above observations that the pre-built models are not sufficiently generalized. It is also difficult to adapt them practically because of poor speed and contain more layers which require more time in execution. Most of the pre-built models are complex in nature and suffer from an overfitting problem with different food classes, while DRCNN runs or generalize well on small as well as large datasets with less or more number of classes.

[2]https://www.kaggle.com/datasets/cdart99/food20dataset

[3]https://www.kaggle.com/datasets/iamsouravbanerjee/indian-food-images-dataset

6.6.3 The Performance Comparison of DRCNN with Other Models on Food-101 Dataset

Food-101 dataset is a widely recognized and commonly used benchmark dataset for image classification tasks in the field of computer vision. The Food-101 dataset contains a large number of images (101,000 images) belonging to 101 different food categories. This diversity makes it suitable for training and evaluating models that need to recognize a wide range of food items.

Many authors have proposed different CNN models to classify food images on the Food-101 dataset. Gozde et al. [160] proposed a Deep CNN (DCNN) model to classify food items for the Food-101 dataset. For this purpose, three different models were proposed and performance was compared. The highest accuracy achieved by DCNN for Food-101 is 77.56%. VijayaKumari et. al. [161] use transfer learning on the existing pre trained model Efficientnet and achieved an accuracy of 80% on the Food-101 dataset. Eduard et al. [162] has proposed a combination of multiple classifiers based on different CNN model that uses different classifiers fusion. The highest classification accuracy achieved for Food-101 is 90.27%. Abdulkadir et al. [163] have fine-tuned the pre-trained Alexnet and VGG16 to classify Food-101 using the SVM classifier. The highest accuracy achieved for Food-101 is 79.86%.

The classification accuracy obtained by the different models proposed for the Food-101 Dataset is shown in Table 6.11.

Table 6.11 Comparative Analysis of DRCNN with other models with Food-101 Dataset

Methods	Accuracy (%)
Proposed DCRNN	98.98
DCNN [160]	77.56
Efficient-Net B0 [161]	80.00
CNN [162]	90.27
CNN [163]	79.86

DRCNN has improved its accuracy by at least 8.71% than the existing DCNN [160], Efficient-Net [161], CNN [162] and CNN [163] models. This indicates that the proposed DCRNN achieved faster and easier training of the network and improved its accuracy than the other methods.

Hence, the analysis concludes that the DRCNN is more efficient and accurate in classifying Gujarati as well as other food images.

Concluding Remarks: A new model from scratch has been developed named DRCNN. The hyperparameters for the model have been chosen using empirical analysis and using Keras Tuner. It achieves a remarkable classification accuracy of 95.48% and a loss rate of 0.8041 for TGFD. The DRCNN size in terms of parameters is 48 times smaller than the prebuilt highest accuracy achiever Inception v3 model.

The accuracy, number of parameters and training time of DRCNN has been compared with all fine-tuned pre-built models. The DRCNN has been tested with a higher number of Gujarati Food items and with different food datasets and it gives remarkable results. The performance of DRCNN for the Food 101 dataset has also been compared with other proposed models by different authors.

The next chapter discusses the time complexity of the CNN model. It will help to decide the crucial parameters for determining the time complexity of the model.