Chapter 5

Simulation and Optimization Using Hyperparameter Tuning

This chapter first explains the simulation done on the pre-built models on the TGFD dataset to analyze the performance of existing models. Then the implementation of transfer learning and hyperparameter tuning on pre-built models to improve the performance on TGFD is discussed.

5.1 SIMULATION

Simulation is an important research method to understand the operational behavior of the model [140]. There are an ample number of pre-built CNN models exist, like VGG16, Alexnet, Xception, EfficientNet and Inceptionv3, which are already trained on the ImageNet dataset and give remarkable performance [140].

In order to check the performance of the pre-built model on the newly created TGFD, the simulation has been performed on some of the networks such as MobileNet, EfficientNet, Xception, LeNet, VGG16, VGG19, Resnet50, Inceptionv3 and Alexnet. The top five networks which gives the highest classification accuracy and minimal loss on TGFD are selected for further improvement and they are: VGG16, VGG19, Resnet50, Inceptionv3 and Alexnet.

Table 5.1 briefly describes top pre-built models of CNN with their main features, error rate, number of parameters used, input size, no. of convolution layers and stride based on recently published papers [34-38]. The detailed architectures of these models have already been discussed in chapter 2, section 2.6.7. In image classification, CNN models assign probabilities to different class labels, indicating the likelihood of an image belonging to each class. The top-5 error rate in the Table 5.1 quantifies the percentage of predictions where the correct label is not among the top 5 predicted labels.

Table 5.1 Classification of CNN

Year	CNN	Main Features	Top 5 Error Rate %	No. of Parameters (in Millions)	Input Size (Pixels)	No. of Convolutional layer	Stride
1998	LeNet [34]	First popular CNN architecture Originally trained to classify handwritten digits	NA	0.06	28x28	2	1
2012	AlexNet [35]	Winner of ImageNet ILSVRC-12, It is deeper as compared to LeNet.	15.3	60	227x227	5	1,4
2014	GoogleNet [36]	Winner of the ILSVRC2014 competition. it has Introduces block concept	6.67	4	224x224	21	1,2
2014	VGG Net [37]	Runner up in the ILSVRC-2014 competition. Homogenous topology Uses small size kernels	7.3	138	224x224	16	1
2015	ResNet [38]	Relu is used Identity mapping-based skip connections and implement heavy batch-normalization	3.6	25.6, 1.7	224x224	50	1,2

5.1.1 Experimental Setup and Result Discussion

The simulation has been done on 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 .

All the five models run from 10 epoch up to 500 epochs. Adam optimizer has been used for the experiments as it gives best classification accuracy, requires less memory, easy to implement and also computationally efficient [153]. The classification accuracy of all the models with the above mentioned parameters is shown in Table 5.2.

Table 5.2 Classification accuracy of models in Simulation

Model Name	Epoch (Accuracy in %)								
	20	40	60	80	100	200	300	400	500
VGG16	70.25	71.89	74.58	77.56	78.24	77.54	77.23	76.25	77.85
VGG19	68.21	70.87	72.55	78.14	80.45	79.87	78.21	80.11	79.54
ResNet50	38.14	40.77	42.88	43.21	45.56	45.87	45.21	45.25	45.21
Inceptionv3	**78.21**	**80.21**	**80.21**	**81.45**	**81.58**	**81.45**	**81.12**	**82.47**	**81.54**
Alexnet	50.21	52.24	55.23	58.54	58.87	58.36	57.45	59.55	58.78

It has been observed from the results in Table 5.2 that accuracy is not improving after 100 epoch and hence, accuracy at 100 epochs has been considered for the comparisons.

The categorical cross-entropy loss function has been used for error calculation as the problem is multi class classification and it computes the average loss across all classes.

The highest accuracy achieved was 81.58% by the Inceptionv3 model, but the classification loss was 20.31%, which is very high. This loss is further increased with the number of epochs, resulting in overfitting the model. ResNet50 does not give good accuracy as the model starts overfitting after 60 epochs. Alexnet fails to achieve good accuracy on TGFD as it is a shallow architecture and since TGFD is a deeper dataset, it requires deep architecture. None of the models provide satisfactory classification accuracy on TGFD.

Increasing model accuracy can be achieved through two distinct approaches: building a model from scratch and leveraging transfer learning or fine-tuning techniques. One strategy to enhance accuracy involves constructing a model from scratch, customize its architecture and training it specifically for the given task or dataset. This approach allows for complete customization and control over the model's design and parameters, enabling optimal performance based on the unique characteristics of the data at hand.

An alternative approach involves utilizing pre-trained models that have been previously trained on extensive datasets for general tasks. Transfer learning involves leveraging the acquired knowledge and representations from the pre-existing models and applying them to tasks or datasets that share similarities. By utilizing this existing knowledge, the model can effectively capture complex features and patterns, resulting in reduced training time and resource utilization. Fine-tuning takes this process further by adapting the pre-trained model to the specific target dataset, allowing for adjustments to be made to particular layers or parameters based on the task requirements.

To increase the accuracy and to see the effect on pre-trained models, transfer learning and finetuning have been implemented on TGFD.

5.2 TRANSFER LEARNING

In transfer learning, as the name suggests, learned features are used from one problem to solve another kind of similar problem with different parameters and different environments in order to increase the accuracy [141]. Transfer learning generally applies

when the new dataset is small as compared to the ImageNet dataset on which all pre-trained models have been trained. The main benefit of using transfer learning is that it speeds up the training time and requires less data.

In CNN upper layers are more generic while the lower layers are more task specific. It is a good idea not to change the upper layers as it has generic features that can be the same for other models and might be trained on different datasets [140-142]. In transfer learning, classification part (lower layers) of the model can be changed, by freezing the rest of the layers to load the trained weights of the model [140-142]. The remaining CNN is now a fixed feature extractor for the new dataset. Fig. 5.1 shows the architecture of Transfer Learning.

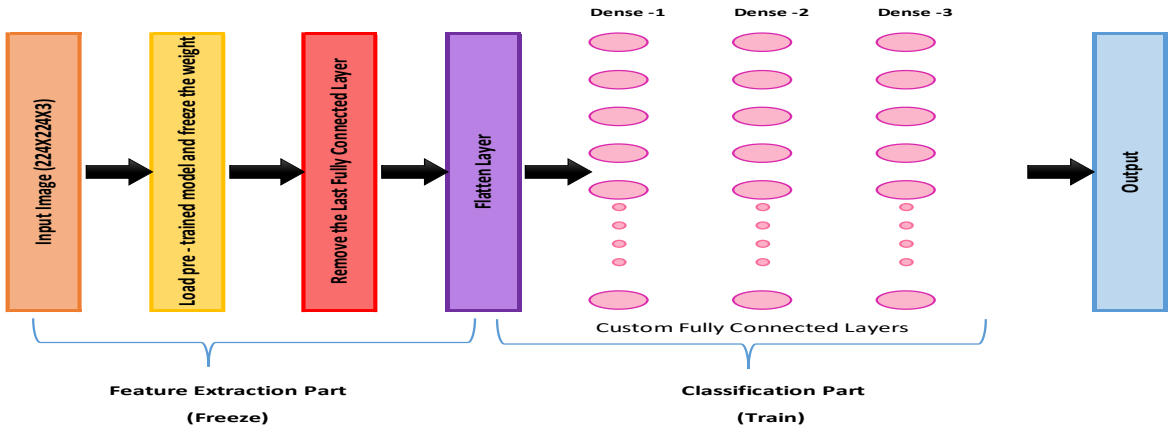

Fig. 5.1 Transfer Learning

5.2.1 Model implementation with Transfer Learning

In this research work, the transfer learning is implemented by changing the classification layer of the model and freezing the rest of the layers with the following parameters.

- A flatten layer followed by a fully connected layer with a Softmax activation function has been added to the model.
- The model is compiled using the Adam optimizer.
- Categorical cross-entropy has been used as a loss function.
- The same environment has been used for all the five models considered.
- The model runs from 20 epochs up to 500 epochs.

After implementing Transfer Learning Fig. 5.2 shows the layers configuration of VGG16, Fig. 5.3 shows the layers configuration of VGG19, Fig. 5.4 shows the layers configuration of Inceptionv3, Fig. 5.5 shows the layers configuration of ResNet50 and Fig. 5.6 Shows the layers configuration of Alexnet.

```
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 5)                 125445
=================================================================
Total params: 14,840,133
Trainable params: 125,445
Non-trainable params: 14,714,688
```

Fig 5.2 Layers Configuration and Summary of VGG16 after Transfer Learning

```
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 5)                 125445
=================================================================
Total params: 20,149,829
Trainable params: 125,445
Non-trainable params: 20,024,384
```

Fig 5.3 Layers Configuration and Summary of VGG19 after Transfer Learning

```
conv5_block3_3_conv (Conv2D)    (None, 7, 7, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

conv5_block3_3_bn (BatchNormal  (None, 7, 7, 2048)   8192        ['conv5_block3_3_conv[0][0]']
ization)

conv5_block3_add (Add)          (None, 7, 7, 2048)   0           ['conv5_block2_out[0][0]',
                                                                  'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)   (None, 7, 7, 2048)   0           ['conv5_block3_add[0][0]']

flatten (Flatten)               (None, 100352)       0           ['conv5_block3_out[0][0]']

dense (Dense)                   (None, 5)            501765      ['flatten[0][0]']

==================================================================================================
Total params: 24,089,477
Trainable params: 501,765
Non-trainable params: 23,587,712
```

Fig 5.4 Layers Configuration and Summary of Resnet50 after Transfer Learning

```
concatenate_1 (Concatenate)     (None, 5, 5, 768)    0           ['activation_91[0][0]',
                                                                  'activation_92[0][0]']

activation_93 (Activation)      (None, 5, 5, 192)    0           ['batch_normalization_93[0][0]']

mixed10 (Concatenate)           (None, 5, 5, 2048)   0           ['activation_85[0][0]',
                                                                  'mixed9_1[0][0]',
                                                                  'concatenate_1[0][0]',
                                                                  'activation_93[0][0]']

flatten (Flatten)               (None, 51200)        0           ['mixed10[0][0]']

dense (Dense)                   (None, 5)            256005      ['flatten[0][0]']

==================================================================================================
Total params: 22,058,789
Trainable params: 256,005
Non-trainable params: 21,802,784
```

Fig 5.5 Layers Configuration and Summary of Inceptionv3 after Transfer Learning

```
activation_23 (Activation)   (None, 4096)              0

dropout_6 (Dropout)          (None, 4096)              0

batch_normalization_21 (Bat  (None, 4096)              16384
chNormalization)

dense_9 (Dense)              (None, 1000)              4097000

activation_24 (Activation)   (None, 1000)              0

dropout_7 (Dropout)          (None, 1000)              0

batch_normalization_22 (Bat  (None, 1000)              4000
chNormalization)

dense_10 (Dense)             (None, 5)                 5005

activation_25 (Activation)   (None, 5)                 0

=================================================================
Total params: 28,084,757
Trainable params: 28,063,621
Non-trainable params: 21,136
```

Fig 5.6 Layers Configuration and Summary of Alexnet after Transfer Learning

5.2.2 Result Discussion

The result of classification accuracy after implementing Transfer Learning on TGFD is shown in Table 5.3

Table 5.3 Classification accuracy of models after Transfer Learning

Model Name	Epoch								
	20	40	60	80	100	200	300	400	500
VGG16	75.56	75.69	77.89	80.23	83.91	83.56	83.36	85.54	84.21
VGG19	80.25	82.56	82.58	84.78	85.06	84.78	83.69	84.78	85.21
ResNet50	45.56	48.89	48.58	50.47	52.3	52.3	52.3	51.78	52.8
Inceptionv3	**80.25**	**84.45**	**85.89**	**86.21**	**86.22**	**86.22**	**85.45**	**85.45**	**85.44**
Alexnet	58.89	60.56	60.56	61.45	62.93	62.93	62.93	61.23	61.25

As seen in Table 5.3, VGG16, VGG19 and Inceptionv3 show steady improvements over the epochs. ResNet50 and Alexnet has less consistent improvement. ResNet50 model starts overfitting after 60 epochs due to the different characteristics and complexities of the dataset. Alexnet is a shallow architecture and lacks the capacity to capture complex features in the Gujarati food dataset and hence, does not perform well for TGFD. The highest accuracy of 86.22% has been achieved by Inceptionv3 model on TGFD.

Fig. 5.7 to Fig 5.11 shows graphs for training and testing accuracy for VGG16, VGG19, ResNet50, Inceptionv3 and Alexnet respectively for TGFD.

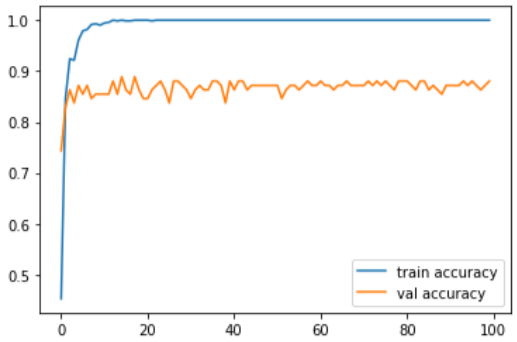

Fig 5.7 Accuracy curve for VGG16 after Transfer Learning

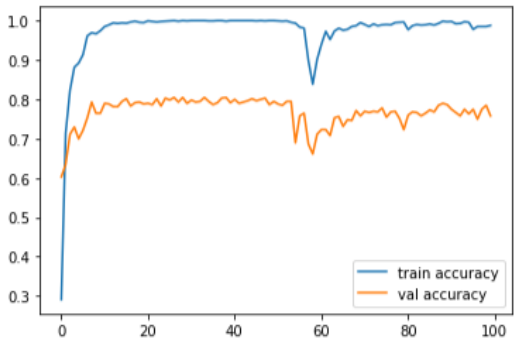

Fig 5.8 Accuracy curve for VGG19 after Transfer Learning

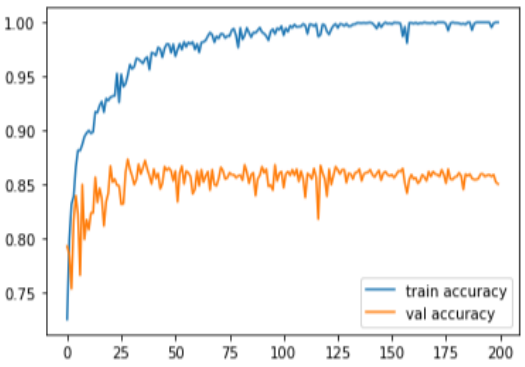

Fig 5.9 Accuracy curve for ResNet50 after Transfer Learning

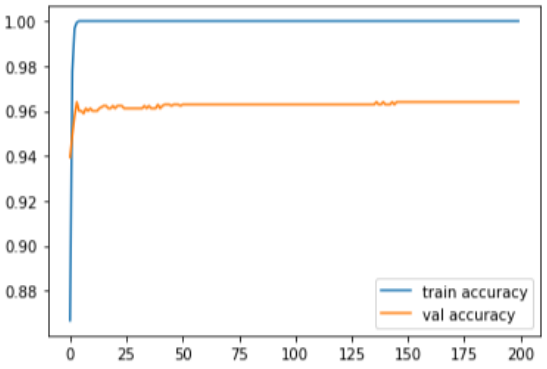

Fig 5.10 Accuracy curve for Inceptionv3 after Transfer Learning

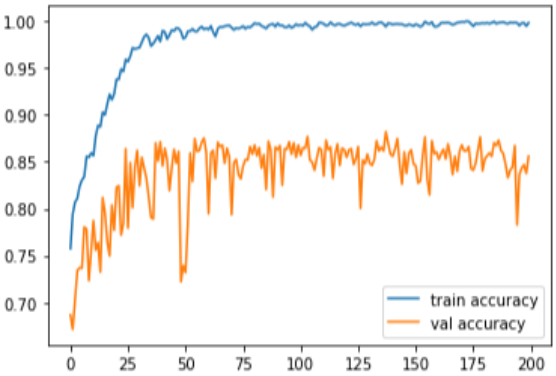

Fig 5.11 Accuracy curve for Alexnet after Transfer Learning

From graphs Fig. 5.7 to Fig. 5.11, it can be seen that there is an improvement in training and validation(testing) accuracy as the number of epochs increases.

Table 5.4 compares the highest accuracy achieved after simulation and transfer learning along with trainable parameters in transfer learning. All the models run 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 5.4.

Table 5.4 Comparison of Simulation and Transfer Learning

Model Name	Simulation Test Accuracy (%)	Transfer Learning Test Accuracy (%)	Trainable Parameters
VGG16	78.24	83.91	1,25,445
VGG19	80.45	85.06	1,25,445
ResNet50	45.56	52.3	5,01,765
Inceptionv3	81.58	86.22	2,56,005
Alexnet	58.87	62.93	21,136

As seen in Table 5.4, ResNet50 does not give good accuracy as the model starts overfitting after 60 epochs. Since TGFD is deeper dataset it requires deeper architecture. Alexnet fails to achieve good accuracy as it is a shallow architecture. The highest accuracy achieved is 86.22% by the Inceptionv3 model. The model contains 256,005 trainable parameters. The accuracy is better than the result of the simulation but not optimal and it takes a large amount of time because of large number of parameters. Since there is still scope for further improvement, fine-tuning has been done on the models to further improve the classification results on the TGFD dataset.

5.3 Hyperparameter Tuning

To further improve the classification accuracy, an effort was made to implement transfer learning along with fine-tuning. Fine-tuning is more flexible as compared to transfer learning as the feature extraction part of the model along with the classification

part can be changed. It is possible to fine-tune all the layers, but it is always a good practice to fix the upper layers as it contains more generic features [46][143]. Fig. 5.12 shows the general architecture of fine-tuning.

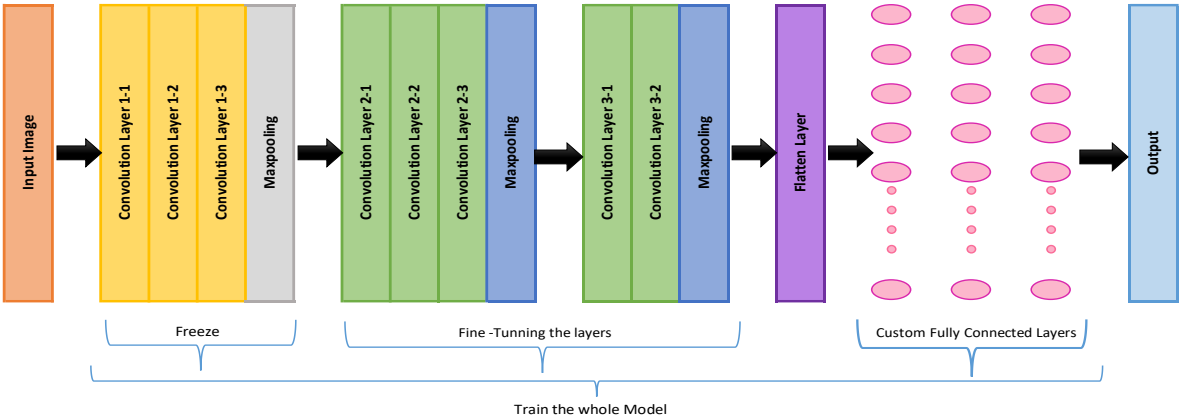

Fig. 5.12. Fine-tuning

5.3.1 Model implementation with Hyperparameter Tuning

In this research work the following changes have been made to implement transfer learning along with fine-tuning. The last Convolutional layer and the pooling layer have been changed, keeping the other layers frozen. A new dense layer along with a dropout has been added [44]. The parameters of the model are as below:

- The dropout rate is set to 0.5 as this gives the best results with hidden layers and dense layers [44].
- When there are more than two classes, Softmax is preferable as it returns probabilities of each class. So in the last layer Softmax activation function has been used.
- The model is compiled using the Adam optimizer.
- A learning rate of 0.002 has been chosen for Adam optimizer.
- The categorical cross-entropy has been used as a loss function as it is a multi-class classification model.
- All the models run starting from 10 epochs to 500 epochs.

After Fine-tuning Fig 5.13 shows the layers configuration of VGG16, Fig. 5.14 shows the layers configuration of VGG19, 5.15 shows the layers configuration of ResNet50, Fig. 5.16 shows the layers configuration of Inceptionv3, Fig. and Fig. 5.17 Shows the layers configuration of Alexnet.

```
block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808
block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0
flatten_3 (Flatten)         (None, 25088)             0
dense_3 (Dense)             (None, 5)                 125445
=================================================================
Total params: 14,840,133
Trainable params: 14,840,133
Non-trainable params: 0
```

Fig 5.13 Layers Configuration and Summary of VGG16 after Fine-tuning

```
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 5)                 125445
=================================================================
Total params: 20,149,829
Trainable params: 20,149,829
Non-trainable params: 0
```

Fig 5.14 Layers Configuration and Summary of VGG19 after Fine-tuning

```
conv5_block3_2_bn (BatchNormal  (None, 7, 7, 512)   2048        ['conv5_block3_2_conv[0][0]']
ization)

conv5_block3_2_relu (Activatio  (None, 7, 7, 512)   0           ['conv5_block3_2_bn[0][0]']
n)

conv5_block3_3_conv (Conv2D)   (None, 7, 7, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

conv5_block3_3_bn (BatchNormal  (None, 7, 7, 2048)  8192        ['conv5_block3_3_conv[0][0]']
ization)

conv5_block3_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block2_out[0][0]',
                                                                 'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block3_add[0][0]']

flatten (Flatten)              (None, 100352)       0           ['conv5_block3_out[0][0]']

dense (Dense)                  (None, 5)            501765      ['flatten[0][0]']

==================================================================================================
Total params: 24,089,477
Trainable params: 24,036,357
Non-trainable params: 53,120
```

Fig. 5.15 Layers Configuration and Summary of ResNet50 after Fine-tuning

```
concatenate_1 (Concatenate)     (None, 5, 5, 768)    0           ['activation_91[0][0]',
                                                                  'activation_92[0][0]']

activation_93 (Activation)      (None, 5, 5, 192)    0           ['batch_normalization_93[0][0]']

mixed10 (Concatenate)           (None, 5, 5, 2048)   0           ['activation_85[0][0]',
                                                                  'mixed9_1[0][0]',
                                                                  'concatenate_1[0][0]',
                                                                  'activation_93[0][0]']

flatten (Flatten)               (None, 51200)        0           ['mixed10[0][0]']

dense (Dense)                   (None, 5)            256005      ['flatten[0][0]']

==================================================================================================
Total params: 22,058,789
Trainable params: 22,024,357
Non-trainable params: 34,432
```

Fig. 5.16 Layers Configuration and Summary of Inceptionv3 after Fine-tuning

```
max_pooling2d_2 (MaxPooling  (None, 1, 1, 256)        0
2D)

flatten (Flatten)           (None, 256)              0

dense (Dense)               (None, 4096)             1052672

activation_5 (Activation)   (None, 4096)             0

dropout (Dropout)           (None, 4096)             0

dense_1 (Dense)             (None, 4096)             16781312

activation_6 (Activation)   (None, 4096)             0

dropout_1 (Dropout)         (None, 4096)             0

dense_2 (Dense)             (None, 5)                20485

activation_7 (Activation)   (None, 5)                0

=================================================================
Total params: 21,601,669
Trainable params: 21,601,669
Non-trainable params: 0
```

Fig. 5.17 Layers Configuration and Summary of Alexnet after Fine-tuning

5.3.2 Result Discussion

The five models VGG16, Resnet50, Inceptionv3, VGG19, and Alexnet have been fine-tuned with above mentioned parameters and their corresponding accuracy for TGFD along with trainable parameters are as shown in Table 5.5.

Table 5.5 Classification accuracy of models after Fine-tuning

Model Name (Trainable Parameter)	Epoch (Accuracy in %)								
	20	**40**	**60**	**80**	**100**	**200**	**300**	**400**	**500**
VGG16	80.21	82.24	84.55	84.55	85.23	84.12	84.32	83.36	84.32
-1,48,40,133									
VGG19	82.21	84.56	85.58	86.69	87.3	87.3	86.32	86.45	86.69
-2,01,49,829									
ResNet50	52.36	55.89	58.45	61.78	62.32	62.32	62.32	61.25	62.32
-2,35,87,712									
Inceptionv3	**82.25**	**85.59**	**86.44**	**86.45**	**89.36**	**89.12**	**88.45**	**88.96**	**88.12**
-2,20,24,357									
Alexnet	60.23	62.45	65.58	66.45	68.73	66.23	67.78	67.98	68.21
-2,80,63,621									

As seen in Table 5.5 VGG16 and VGG19 both have improvement in accuracy however VGG19 achieves higher accuracy than VGG16.ResNet50 and Alexnet have lower accuracy as compared to other models although it shows some improvement in initial epochs. It indicates that the architecture and capacity of both the models may not be suitable for TGFD.Inceptionv3 shows consistent improvement in accuracy as the number of epochs increases. This suggests that it is effective at learning complex features presents in the TGFD.

Fig. 5.18 to Fig. 5.22 shows graph for the training and testing accuracy curve for VGG16, VGG19, ResNet50, Inceptionv3 and Alexnet respectively.

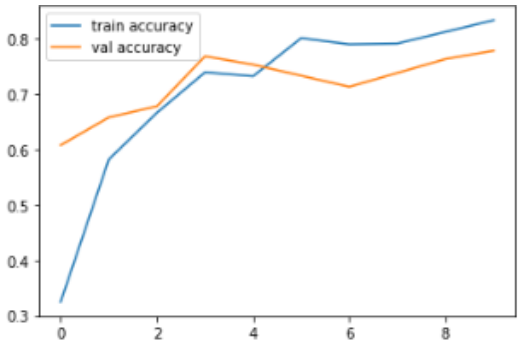

Fig. 5.18 Accuracy curve for VGG16 after Fine tuning

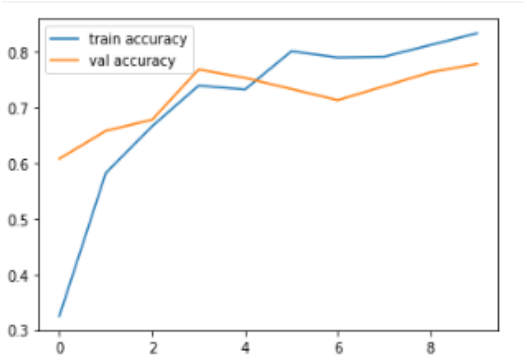

Fig. 5.19 Accuracy curve for VGG19 after Fine tuning

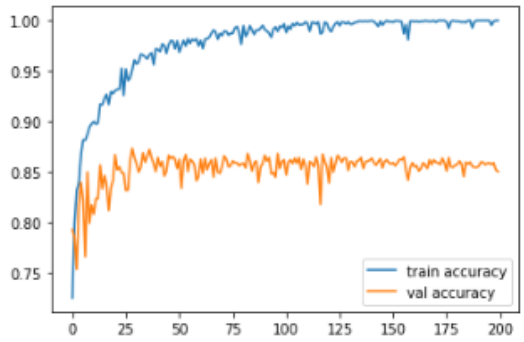

Fig. 5.20 Accuracy curve for ResNet50 after Fine tuning

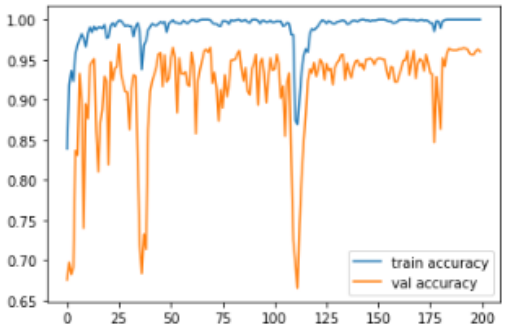

Fig 5.21 Accuracy curve for Inceptionv3 after Fine tuning

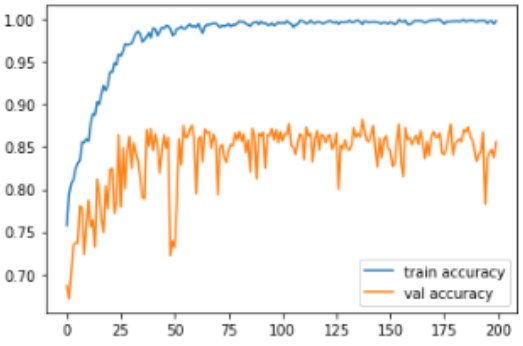

Fig 5.22 Accuracy curve for Alexnet after Fine tuning

From Fig. 5.18 to Fig. 5.22 it has been observed that training accuracy is more than validation(testing) accuracy for all the models. Table 5.6 shows the comparison of classification accuracy of simulation, Transfer Learning and Fine-tuning for 100 epochs.

Table 5.6 Comparison of classification accuracy of Simulation, Transfer learning, Fine-tuning

Model Name	Simulation Test Accuracy (%)	Transfer Learning Test Accuracy (%)	Fine-tuning Test Accuracy (%)
VGG16	78.24	83.91	85.23
VGG19	80.45	85.06	87.3
ResNet50	45.56	52.3	62.32
Inceptionv3	81.58	86.22	**89.36**
Alexnet	58.87	62.93	68.73

As seen in Table 5.6, Simulation, transfer learning and fine-tuning have been implemented for five models, namely VGG16, VGG19, Inceptionv3, Alexnet and Resnet50. The best classification accuracy achieved by Inceptionv3 after transfer learning is 86.22% and after fine-tuning, it is 89.36%. The models performance is consistently improving from simulation to transfer learning and from transfer learning to fine-tuning. By implementing transfer learning and fine-tuning, the testing accuracy of models has been increased by at least 5% and 8%, respectively, compared to simulation. It proves that transfer learning along with fine-tuning significantly improves the classification accuracy.

The graphical presentation of comparison of classification accuracy of simulation, transfer learning and fine-tuning for all the five pre-built models have been shown in Fig. 5.23.

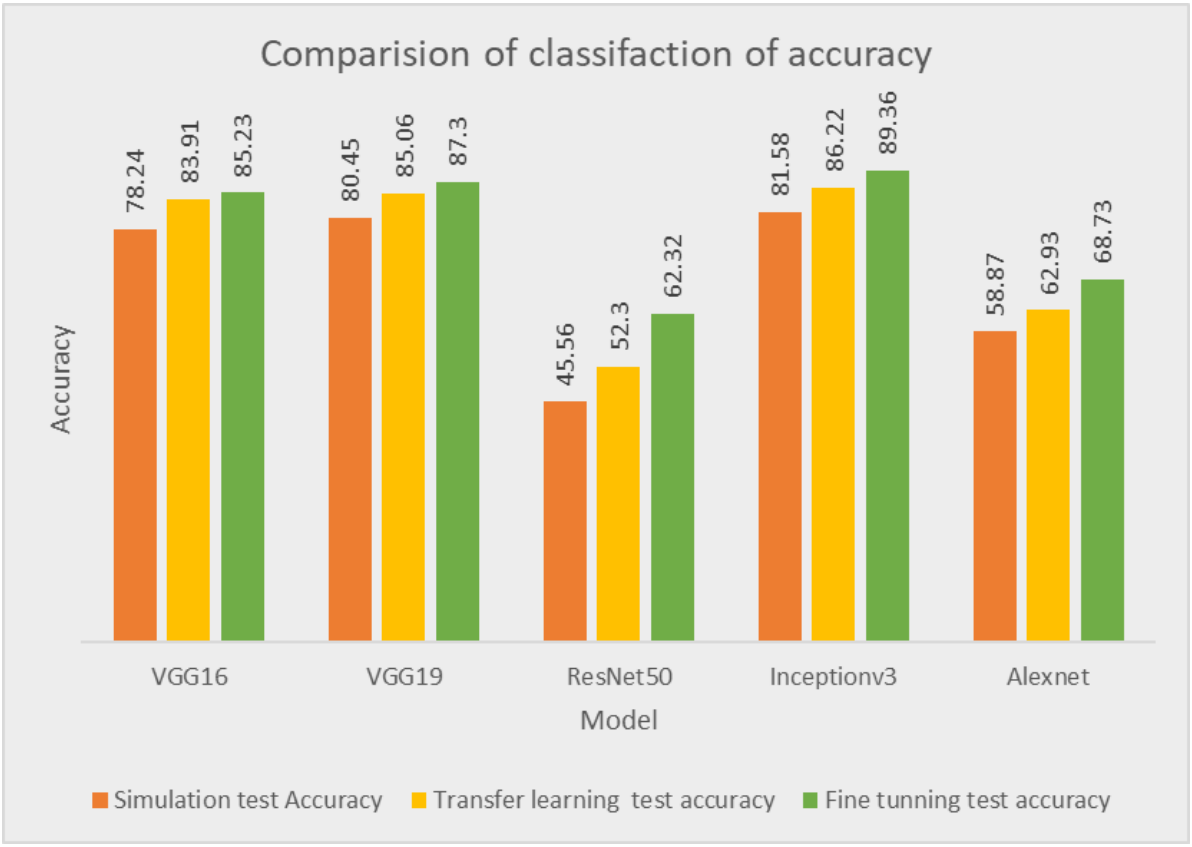

Fig 5.23. Comparison of classification accuracy

As TGFD is a deeper dataset and according to the research findings, a deeper network gives better results with a deeper dataset. As Alexnet is shallow architecture it doesn't give a good result with TGFD. Resnet50 model begins to overfit after 60 epochs as it has been trained on very large-scale dataset. From the experiments it has been clear that all the pre-built models are not able to learn unique visual features or complex features of Gujarati food. It shows model's architecture is not suitable for the specific features of Gujarati food.

To resolve the issue of overfitting and to further improve the classification accuracy, a model from scratch has been developed.

Concluding Remarks: A simulation has been done on five pre-built models to check their performance on TGFD. To improve the accuracy achieved in simulation, transfer learning has been implemented on five models: VGG16, Resnet50, Inceptionv3, VGG19 and Alexnet for TGFD.

Fine-tuning is more flexible than transfer learning hyperparameter tuning has been implemented on all these five models. The accuracy achieved by simulation, transfer learning and fine tuning has been compared.

To further increase the classification accuracy and reduce the learnable parameters a lightweight model has been proposed which will be discussed in the next chapter