Chapter 3

CNN-based recognition of TUMMHCD

This chapter first presents a performance analysis of some recent state-of-the-art CNN models on TUMMHCD. It identifies their drawbacks as far as the present dataset is concerned. A CNN model is then built from scratch for the recognition of TUMMHCD. Experimental results show a comparison of the state-of-the-art models and the proposed model.

3.1 Convolutional neural network (CNN)

Convolutional Neural Network is a type of artificial neural network whose unique architecture and working is proven best for image classification and computer vision tasks in which the input images are two-dimensional. Much like traditional Neural Networks, CNNs consist of neurons with trainable weights and biases. However unlike the ordinary NNs, the structure of CNN makes it possible for this particular variant of NN to deal with much fewer parameters and to share parameter weights. The architecture of a CNN imitates the pattern in which neurons are connected in the human brain and is analogous to the organization of animal visual cortex [62, 63, 80]. Individual neurons react to stimuli only in a defined portion of the visual space. This defined portion is known as the receptive field. The receptive fields of different neurons partially overlap to cover the entire visual area. Receptive field of a layer is defined by the filter size of that layer within a CNN. Benefit of receptive fields in recognizing visual pattern is that the neurons in a particular layer are assigned to learn visual features from a small area

of the input image. The scope of a neuron within a layer on the input data is only within this small area called the receptive field. Moreover, each filter shares the same weights. This makes the number of trainable parameters in CNNs to reduce considerably. This is not the case with traditional NNs where every neuron within a layer is connected to every other neuron in the previous layer.

Another advantage of using CNNs is that they preserve the spatial relationship between pixels in an image. This is not the case with traditional feedforward NNs where the image pixels are flattened into a long vector of pixel values leading to loss of spatial information present in the image. It is very important that the spatial relationship of pixels in an image be kept preserved as the pixels close by are spatially more co-related to each other than the pixels which are far away from each other. This task is achieved in CNNs by learning internal feature representations using the receptive fields. Features learned are used across the entire image meaning that it can detect the same feature anywhere in the entire input space. For example, lower layers of CNNs learn low-level features such as edges, lines, etc. The presence of edges and lines anywhere in the image can therefore be detected using the same feature representation. This very property of preservation of spatial information in CNNs makes it a powerful tool for computer vision tasks.

Benefits of using convolutional neural networks is summarized below:

- Less number of parameters (weights) are required to learn than a fully connected network.
- They are designed to be invariant to translation and rotation of the image.
- They learn features automatically from the input data and hence are independent of human intervention for feature extraction.
- Through automated learning, CNNs learn to optimize the filters and hence they need relatively little pre-processing of the input data.

A typical CNN consists of three main layers viz. convolutional, pooling and fully connected layers which are assembled one after another. The number of layers and the manner in which they are stacked is really a design issue which the network architecture designer has to decide upon. The three basic layers of a CNN are described below.

Convolutional layer

Convolutional layer is the first layer and is the core building block of a CNN. As the name suggests, *convolution* operation is performed in this layer. The convolution is between a defined area of input image (receptive field) and a two-dimensional array of weights known as filter/kernel. In regard to CNNs, a convolution operation is the element-wise multiplication between the defined area which is the filter-sized region of the input space and the filter. Convolution is carried out systematically on every filter-sized patch of the input image by moving left to right with a defined stride value till it covers the entire width of the input image. It then moves down to the beginning of the image with the same stride value and continues till the entire width of the image is parsed and so on until the whole width and height of the image is covered. Each convolution of the filter with an input patch produces a single value. When this operation is carried out multiple times using the same filter over all the possible patches of the input image, a two-dimensional array is produced. This two-dimensional output array is called an activation map or a feature map. Every filter is applied across the whole image. This allows a filter to detect a specific feature anywhere in the entire input image. The network therefore learns filters that activate when specific type of features are detected at certain spatial position of the input image. This is the most important idea behind convolutional layer. Feature maps for all filters are then stacked to form the output of a particular convolutional layer. These feature maps act as input for the next layer in the network.

Figure 3-1 shows an example of a filter undergoing convolution operation with an input image to produce a feature map. The filter size is 3×3 and input image size is 5×5. The stride value taken is 1 with no padding of the input image. The dashed lines and dotted lines indicate two applications of the filter onto the input image. The square in bold on the input image is the receptive field and is of the size of filter which is 3×3. Convolution of the filter over the entire input image produces a 3×3 feature map. If another convolutional layer follows this layer, then convolution is carried out on the output feature maps of the present convolutional layer and so on. Once a feature map is obtained, it is passed through a nonlinear function such as Sigmoid function or Rectified Linear Unit (ReLU).

Let l and m be the input and output feature map sizes respectively, k is the filter size, p is the padding size and s is the stride. λ and β be the number of input and output feature maps respectively. And γ be the number of channels of the input image. Then the output feature map size is calculated using the

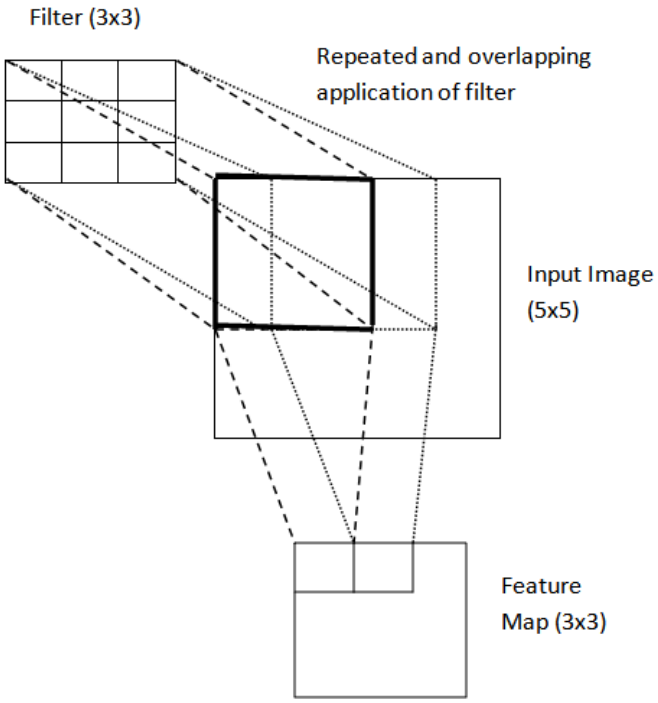


Figure 3-1: Convolution operation of stride 1 between a filter and an input image to produce a feature map.

following formula:

$$m = \left\lfloor \frac{l - k + p}{s} \right\rfloor + 1 \tag{3.1}$$

The number of connections n can be represented mathematically by the below formula:

$$n = ((m * m * (k * k * \gamma) + 1) * \beta \tag{3.2}$$

where 1 is the bias. Similarly, the number of trainable parameters δ is given by the formula:

$$\delta = (k * k * \lambda + 1) * \beta \tag{3.3}$$

Pooling layer

The pooling layer generally follows a series of one or more convolutional layers. This layer performs down sampling of its input data. Input to this layer is the output of the previous layer, i.e the feature maps. Although convolution facilitates in the detection of features anywhere in the input image, the issue with output feature maps is that they are sensitive to the position of the features. This sensitivity can be addressed by the process of down-sampling the feature maps. Down-sampled feature maps reduces sensitivity towards change in the location

of features in the input image. Pooling is carried out in patches of the feature maps. The output of pooling layer or down-sampled feature maps makes the representation invariant to translations and rotations of the input image [67]. The resulting robustness the pooling layer brings about is termed as *local translation invariance*. There are two common types of pooling: average pooling [34] and max pooling [138]. Average pooling takes the average value of each patch on a feature map and max-pooling takes the maximum value. Figure 3-2 shows an example of max-pooling and average-pooling. A filter of size 2×2 is used with stride 2.

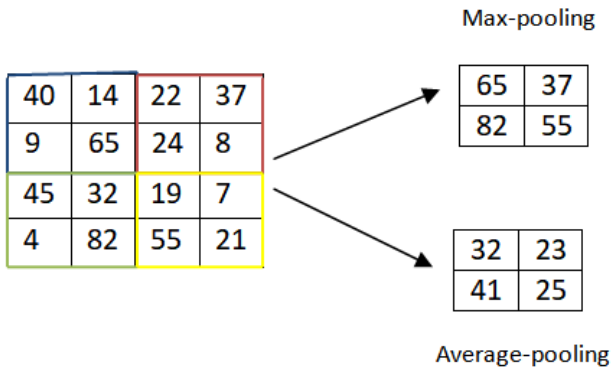


Figure 3-2: Max pooling and average pooling

Fully-connected layer

One or more fully-connected layers come at the end of a CNN architecture. This layer is the classification layer which works in the same manner as the traditional feedforward neural network layer. The extracted features are flattened to a column vector and fed to the classification layer. Learning by the network takes place by applying backpropagation over every iteration of training phase. They have a non-linear activation function or a softmax function to predict the classes of input images.

3.2 Related work

The essence of CNN was felt when Fukushima et al. [61] introduced a Neural Network model called ”neocognitron” in the year 1988. It is a multilayered cascaded network having several connections of layers of cells. The catch was that

these connections could be updated by learning and the weights could be optimized. However, due to difficulty in the learning algorithm it could not be used to its full potential. Later in 1998, Yann LeCun et al. [111] popularized CNN with the introduction of backpropagation and gradient-based learning to train a Neocognitron-like architecture. The CNN model was used to recognize digits in the famous MNIST dataset. Since then, many researchers have worked on the same dataset by proposing and adopting different techniques on the base CNN concept and achieved record-breaking results almost each time. The success of CNNs on this particular dataset can be seen at http://yann.lecun.com/exdb/mnist/. Even though there have been record-breaking results on MNIST, works are still being carried out on major world scripts to improve the recognition accuracies.

CNNs have been explored for HCR of various scripts such as Chinese [40, 41, 132, 234], Arabic [18, 19, 56], Hangul [96]. A number of HCR systems also achieved superior results by enhancing in terms of the architecture and employing additional techniques on the concept of CNN. Fusion of features from different layers of CNNs have been reported in some works [14, 59, 140].

3.2.1 CNN for HCR of Indic scripts

There are some very recent works in the literature on Indic scripts using CNN. The work of Rahman et al. [163] was one of the firsts in HCR of Bangla. They reported a recognition accuracy of 85.96% on a 50-class Bangla dataset by employing a simple CNN model. Alom et al. [13] used CNN with Gabor features and dropout and achieved a recognition accuracy of 98.78% on Bangla numeral dataset. A multi-scale multi-column CNN architecture was proposed by Sarkhel et al. [188]. They tested their methodology on nine publicly available Indic script datasets viz. Bangla, Tamil, Telugu, Urdu and Devanagari. The work of Roy et al. [175] developed a supervised layer wise training of a deep CNN for recognition of Bangla compound characters. A three-column three-level multi-scale skip-connected CNN is proposed in the study carried out by Singh et al. [199]. Significant results were reported using this methodology on Bangla handwritten character datasets. More works on CNN-based HCR of different Indic scripts such as Malayalam [90, 124, 125, 157], Devanagari [2, 49, 71, 166], Tamil [93, 119, 161], Telugu [15, 137], Kannada [167–169], etc. are reported.

3.2.2 CNN for HCR of Meitei Mayek

The first work reported on Meitei Mayek HCR based on CNN adopted a simple CNN model [76]. The authors achieved a recognition accuracy of 96.24% on a 37-class dataset. They could enhance the accuracy to 97.09% after employing data augmentation. Other works are carried out by Inunganbi et al. [85] and Hazra et al. [73] where recognition accuracies of 99.02% and 99.27% were achieved on a 27-class Meitei Mayek dataset. The effect of handcrafted features when fed to a CNN has been studied in the work reported by Nongmeikapam et al. [144]. They have considered HOG feature images to train a CNN and to extract more refined features for recognition of 56 characters of Meitei Mayek. The HOG-CNN features are then used with KNN classifier. They found that HOG-CNN features with KNN gives the best test accuracy compared to handcrafted features with SVM or features learned by CNN only. The authors achieved 98.71% accuracy on a 56-class dataset.

3.3 Performance analysis of state-of-the-art CNN models on TUMMHCD

Performance analysis of five recent state-of-the-art CNN models have been considered in the present work. The CNN models have shown very good results on the ImageNet dataset on which the famous ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is also carried out [178]. It is a large dataset of annotated photographs intended for computer vision research [48]. Transfer learning with fine tuning the five CNN models pre-trained on the ImageNet dataset for the recognition of TUMMHCD has been studied. In total, the ImageNet dataset has around 14 million images, more than 21,000 classes and more than 1 million images with bounding box annotations. The dataset used for image classification comprises 1.2 million images for training, and 50,000 for validation, from 1,000 classes. It has been developed and maintained by researchers in Stanford, Princeton and other American universities.

The five CNN models are briefly described below:

- *InceptionV3 (2015)* [207]: InceptionV3 is CNN model developed by Google after its two previous variants viz. InceptionV1 (GoogleNet) and InceptionV2. It is popular because of its computational efficiency and fewer pa-

rameters compared to earlier models like Alexnet and VGG with a lower error rate. It achieved a top-1 accuracy of 77.9% and top-5 accuracy of 93.7% on ImageNet validation dataset.

- *ResNet50V2 (2016)* [75]: Resnet comes from Microsoft. The main contribution of ResNet architectures is its technique to reduce vanishing gradient problem in deep neural networks. The idea is to introduce "residual blocks" which have "skip connections" where there are direct connections between layers by skipping some layers in between. The skip connections allow an alternate shorter path for the gradient to flow through thereby solving the problem of vanishing gradient. ResNet50V2 achieves top-1 and top-5 accuracies of 76.0% and 93.0% respectively on ImageNet validation dataset.

- *DenseNet121 (2017)* [79]: DenseNets also aim to fight the vanishing gradient problem in deep neural networks by simplifying the connectivity pattern between the layers. They do so by allowing maximum gradient flow betweeen the layers by directly connecting each layer with all other deeper layers. This way, they require fewer parameters to learn and avoid learning redundant feature maps. Because of its very narrow layers with a smaller number of filters, the layers add only a small number of new feature maps. DesneNet121 achieves a top-1 accuracy of 75.0% and a top-5 accuracy of 92.3% on ImageNet validation dataset.

- *MobileNetV2 (2018)* [187]: It is another recent architecture laid out by Google. It is built upon its earlier variant called MobileNetV1. The main aim of this architecture is to reduce the complexity cost and depth of network for the benefit of devices with low computational power like mobiles devices while giving better accuracy. The authors could achieve this by introducing two new features on top of the depth wise separable convolution introduced in MobileNetV1. The two new features are 1) linear bottlenecks between the layers, and 2) shortcut connections between the bottlenecks. It achieves 71.3% top-1 accuracy and 90.1% top-5 accuracy on ImageNet validation dataset.

- *EfficientNetB3 (2019)* [208]: EfficientNets are the latest CNNs architecture from Google. The authors proposed that both accuracy and computational efficiency could be achieved by adopting similar architectures. They claimed that a common architecture with small values of width, depth and resolution can create a computationally efficient model and by keeping these parameters bigger, a better accuracy could be achieved. They are currently the best performing CNN models for image classification tasks under varying

resource availability situations. EfficientNetB3 achieves top-1 accuracy of 81.6% and top-5 accuracy of 95.7% on ImageNet dataset.

Transfer learning: It is the process where what has been learned for one problem is exploited to improve generalization in another problem [67]. In transfer learning, a base network is first trained on a base dataset and the learned features are then transferred to a new target network. The new target network is then trained on a target dataset using the transferred learned features of the base network [238]. Pre-trained models are also available which are already pre-trained on a base dataset and are available for use for our target datasets. In the present work, pre-trained models of the five aforementioned CNNs have been used for transfer learning to analyse their performances against TUMMHCD. The models are pre-trained on ImageNet dataset.

For transfer learning, layers from the pre-trained model except for the batch-normalization layers are frozen. The top fully-connected layer of the pre-trained base model is removed and new top layers are added. We have added a global average pooling layer, a dropout layer of 0.2 and output layer of 55 nodes since our dataset has 55 character classes. We then train the base model with newly added top layers on our dataset. We have also carried out fine-tuning as our dataset has significant number of samples in each class. For fine-tuning, some of the top layers of the base model are unfrozen and the new model is re-trained on the new dataset.

Training the pre-trained models: For training and validation purpose, training set of 72,330 images is divided randomly into training set and validation set in the ratio 9:1. Training set and validation set thus consist of 65,097 and 7,233 character images respectively. In deep neural networks, one of the main challenges while training is the problem of overfitting or overtraining the network. This means that when a neural network is overtrained with the training set with more number of epochs than what is required, it tends to suffer from overfitting. Overfitting is undesirable as it causes the network to overfit the training data and will perform poorly with the unseen data or the test data. On the other hand, undertraining the network is again undesirable as it might result in an underfit model. In order to reduce this problem of overfitting or underfitting, we have used the early stopping method while training the CNN. The challenge is thus to train the network long enough on the training set but halt the training when its performance on the validation set begins to degrade. Early stopping achieves this task by allowing us to set an arbitrarily large number of epochs for training the network but letting the training to stop when the validation loss starts to increase

or validation accuracy starts to decrease.

For the present work, the validation loss is considered as the criterion for monitoring the model performance. The validation loss is monitored for a set of 10 consecutive epochs each time. When the validation loss does not decrease for a consecutive 10 epochs, then the model weights for the epoch with minimum validation loss are saved as the best model and the training stops. If, however the validation loss decreases at a certain epoch during this 10-epoch monitoring, then that particular epoch is set as the first epoch for the next 10-epoch set and so on. This process continues until the epoch with minimum validation loss is found else the training continues for a maximum of 30 epochs. Once this best model is found, it is used to test the accuracy on the test set. Adam optimizer is used for all the models with initial learning rate of 0.001 during the first 20 epochs. The learning rate is reduced to 0.00001 for fine-tuning part. For each model, the corresponding pre-processing is applied to the images in dataset before feeding them to the models.

Observations based on performance of pre-trained CNN models: The experiments are carried out on Google colab with an allocated RAM size of 13GB and 2.20GHz Intel(R) Xeon(R) CPU. The results obtained from transfer learning of the five CNN models on TUMMHCD are given in Figure 3-3. It can be observed from the figure that transfer learning and fine tuning need a lot of resources for large datasets such as ours. Except for MobileNetV2, the amount of time taken for training and testing is huge for the other four models. It also shows that deeper models do not always give better accuracy. Up-scaling the input-image size increases memory usage during training for large neural networks. For some pre-trained models, there is a minimum input size of the images which should be used in order to use the model. For example, the minimum input size of InceptionV3 pre-trained model is 75×75 which increases the requirement of resources during training. Therefore, there is an intuition that simpler models with low resource requirements can be built which might perform equally well or better than deeper networks. A CNN model is thus built from scratch for recognition of TUMMHCD. The details are given in the next section.

CNN model	Input image size	Training time/epoch(s)		Validation accuracy		Total no. of layers in base model	Layer fine tuning starts	Test accuracy		Testing time(s)	
		TL	FT	TL	FT			TL	FT	TL	FT
InceptionV3	75x75	3844	3825	91.9	91.90	311	250	92.22	92.56	83.27	92.77
ResNet50V2	32x32	3768	4134	83.9	88.49	190	140	84.13	88.49	82.68	82.77
DenseNet121	32x32	994	826	16.6	91.15	427	370	17.18	90.81	83.98	81.96
MobileNetV2	32x32	447	465	79.9	93.51	154	100	79.47	93.49	10.75	10.26
EfficientNetB3	32x32	1523	1574	94.64	95.41	384	330	94.36	95.21	43.17	37.79

Figure 3-3: Performance of state-of-the-art pre-trained CNN models against TUMMHCD using transfer learning (TL) with fine tuning (FT).

3.4 Building CNN from scratch for recognition of TUMMHCD

3.4.1 Base CNN architecture

In the present study, a base architecture of CNN inspired by the work of Yann LeCun et al. [111] is considered to have an idea of how hyper-parameters should be tuned to achieve a good test accuracy. It consists of seven layers as shown in Figure 3-4. For implementation of CNN models, Keras API [38] is used on top of Tensorflow [1] software library.

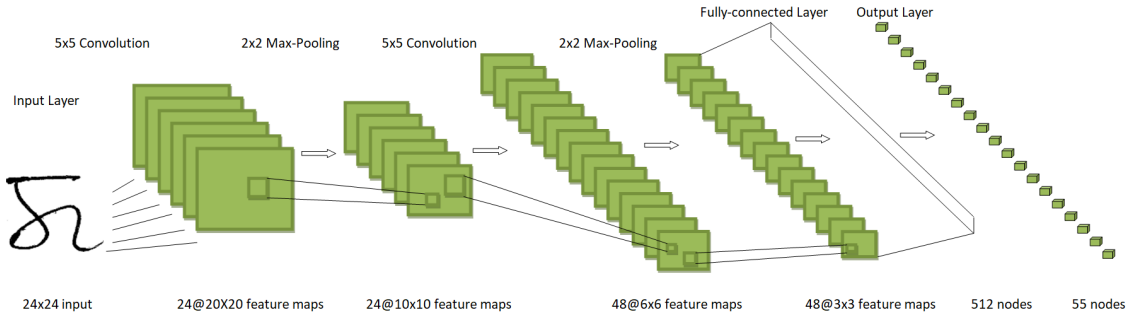


Figure 3-4: Base CNN architecture used for hyper-parameter tuning

For the base architecture, gradient descent optimization algorithm with mini-batch is used with a momentum of 0.9. It is a type of the gradient descent algorithm where the gradient can be made to sum over mini-batch which further reduces the variance of the gradient. It brings about a more robust convergence than batch gradient descent as model update frequency is higher and thus prevents local minima. Rectified Linear Unit (ReLU) activation function is used for the two convolutional layers. ReLU prevents deep learning networks from the problem of vanishing gradient as it squeezes its input x into a broader range of $max(0, x)$ [139] unlike functions like tangent $(f(x) = (1 + e^{-x})^{-1}))$ and sigmoid $(f(x) = tanh(x))$

which map their input onto a very narrow range of output values. For the output layer softmax activation function is adopted with cross entropy loss function.

Pre-processing: Pre-processing is employed to bring the images to a format which further stages can process. One of the advantages for using CNN as a recognition tool is that the input to such a system does not need a lot of pre-processing as compared to that needed for hand-crafted feature extraction. In the present study, the only pre-processing that has been done is normalization of the raw gray-scale images to the pixel intensity values in the range [0, 1]. These normalized images are then fed into the system.

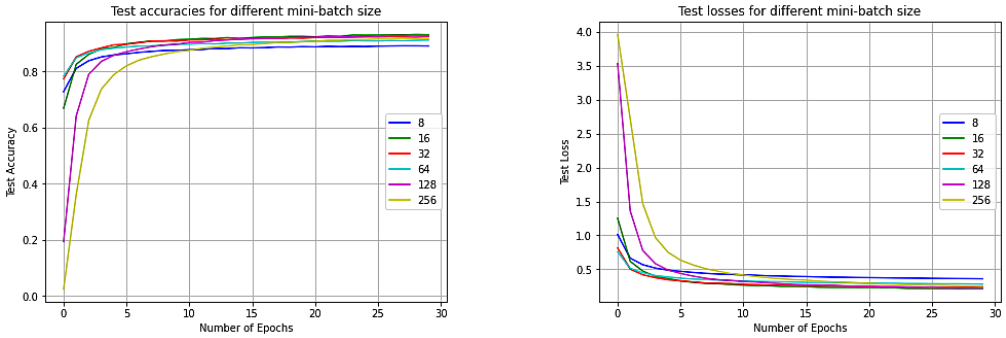
1. Hyperparameters Tuning: A hyperparameter for a learning algorithm is a parameter whose value is used to control the learning process. The task of tuning them to achieve optimal values is a challenging one. For a system that uses stochastic gradient descent optimization, the two hyper-parameters whose fine tuning affects the test accuracy most are batch size and learning rate [26]. Mini-batch size, initial learning rate and learning rate decay are the three hyperparameters we have fine-tuned for our model. We have considered test accuracy to be the basis for selection of hyperparameter values.

    - Mini-batch size: Mini-batch sizes of 8, 16, 24, 32, 128 ad 256 are considered with an initial learning rate (α) of 0.01 which is reported to be the best default value [26] and decay after each iteration i given by

    $$\alpha_i = \frac{\alpha}{\eta} \tag{3.4}$$

    where η is the mini-batch size. Test accuracies and losses using different mini-batch sizes are shown in Figure 3-5. It can be seen in the figure that a mini-batch size of 32 achieves higher test accuracy with less number of epochs. Hence this batch size is considered for the present work. Results of the works in [26] and [130] also suggest mini-batch size of 32 to be a good default value.

    - Initial Learning Rate: The learning rate may be considered as the most important hyperparameter that needs to be tuned in a deep learning network [84]. It is the amount by which the weights should be changed in accordance with the estimated error for each weight update while training the network. It is a difficult task as a value too large might cause the network to converge too quickly leading to sub-optimal solution whereas a learning rate too small may cause the training process

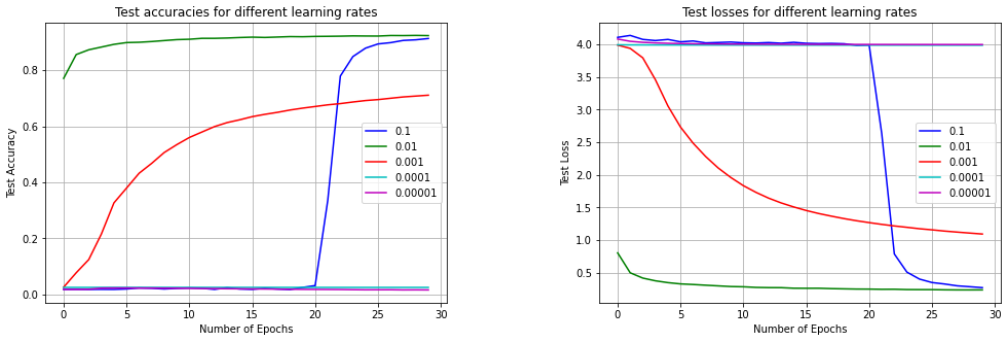


(a) **(b)**

Figure 3-5: (a) Accuracies and (b) Losses using different mini-batch sizes

to become very slow and to get stuck. However, it is not possible to determine the optimal learning rate a priori [171]. It has to be calculated via hit and trial. For our work, initial learning rates of 0.1, 0.01, 0.001, 0.0001 and 0.00001 are considered which give the results shown in Figure 3-6.

It can be observed from Figure 3-6 that an initial learning rate of 0.01 results in fastest convergence and highest test accuracy within the maximum 30 epochs considered in the experiment and hence this value is taken.

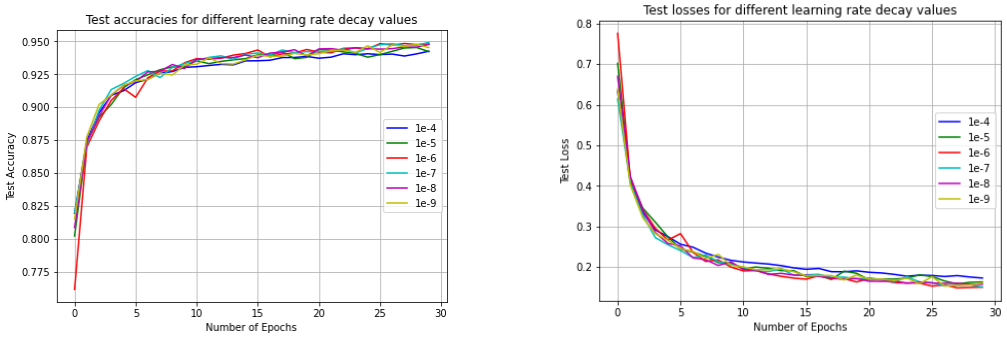


(a) **(b)**

Figure 3-6: (a) Accuracies and (b) Losses at different learning rates

- Learning Rate Decay: It is the amount by which learning rate is decreased over time during training. This is done to avoid larger steps as learning convergences towards the optimal solution as larger steps may lead the learning to never really converge or have a noisy convergence in the region of minima. However, if a smaller learning rate is taken as learning converges, we are making sure that the steps we take when learning converges are small and hence be able to converge to a value close to minima.

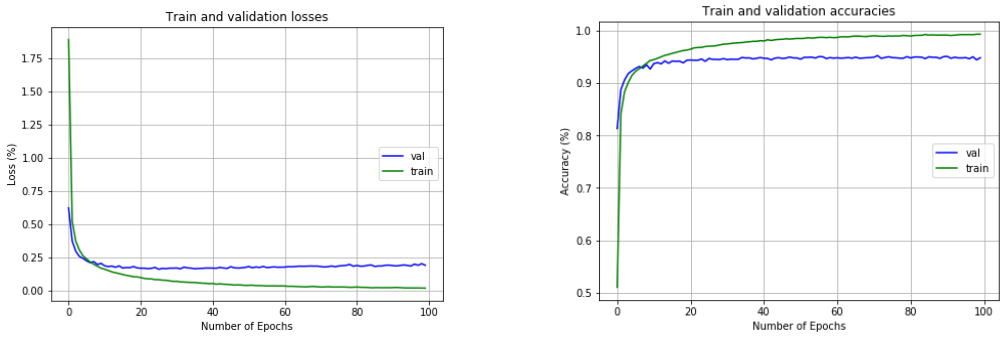
Test losses and test accuracies using learning rate decay values of $1e-4$, $1e-5$, $1e-6$, $1e-7$, $1e-8$ and $1e-9$ are shown in Figure 3-7. Learning decay rate of $1e-7$ is considered for our work.



(a) **(b)**

Figure 3-7: (a) Accuracies and (b) Losses at different learning rate decay values

With the base CNN architecture and the specified hyper-parameters, we could achieve a highest test accuracy of 94.36% when the model is trained for 100 epochs. Figure 3-8 shows the train and test losses and accuracies achieved by adopting the base CNN model on the concerned dataset.



(a) **(b)**

Figure 3-8: (a) Losses and (b) accuracies using base CNN model

3.4.2 Proposed CNN architecture

The overall structure of the proposed CNN architecture is shown in Figure 3-9. It consists of nine layers excluding the input layer. There are four convolutional layers, two max-pooling layers, two fully-connected layers and the output layer. The architecture has the following order: Input-C_1-ReLU-C_2-ReLU-BN-P_1-C_3-ReLU-BN-C_4-ReLU-P_2-FC_1-FC_2-Output. C_i, P_i, FC_i denote the i^{th} convolu-

tional, max-pooling and fully-connected layer respectively. ReLU denotes Rectified Linear Unit (ReLU) activation function and BN denotes batch normalization layer. Max-pooling is employed instead of average-pooling as it can lead to faster convergence, extract invariant features more efficiently and improve generalization [190]. ReLU activation function is adopted as it is shown to be computationally faster and reduces the likelihood of vanishing of the gradient unlike sigmoid and tangent functions [139]. Mini-batch stochastic gradient descent with momentum is used as it converges faster than the original or batch gradient descent [114].

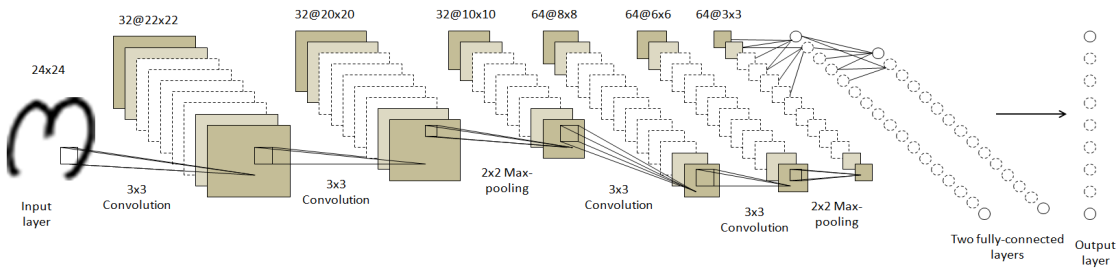


Figure 3-9: Overall structure of the proposed CNN architecture. $x@y \times y$ indicates x output feature maps of size $y \times y$. $z \times z$ Convolution or Max-pooling indicates convolution or max-pooling with kernel masks of size $z \times z$ respectively.

The first convolutional layer (C_1) has an output mapping of 32 feature maps produced by convolution of the input image with filter mask of size 3×3 with stride 1. The activation function used is ReLU (Rectified Linear Unit). The feature maps at this layer is of size 22×22 (24-3+1) pixels. The expressions given inside the brackets are the calculations behind the obtained figures (refer equations 3.1, 3.2 and 3.3) The total number of connections at this layer is 1,39, 424 ((22*22*9+1)*32). However, the number of trainable parameters is very less compared to this figure because of the weight-sharing feature of CNN. The number of trainable parameters at this layer is 320 ((3*3+1)*32). The second convolutional layer (C_2) produces 32 feature maps of size 20×20 (22-3+1) after convolution operation with filter size of 3×3. The number of trainable parameters at this layer is 9,216 ((3*3*32)*32). In this convolutional layer and subsequent convolutional layers, bias is not used as the convolutional layers are followed by batch normalization layer which shifts the activation by their mean values. Hence, it does not make sense to add another bias term in convolutional layer.

Next layer is the first pooling layer (P_1) which is responsible for CNN being able to achieve local-distortion and translation invariance to a certain extent and it also makes the system robust to noise. In the present architecture, max pooling is employed with sub-sample size of 2×2 pixels with stride 2. This layer produces down-sampled output of 32 feature maps with size 10x10 pixels. The number of

trainable parameters is 0.

The third convolutional layer (C_3) gives out 64 feature maps of size 8×8 (10-3+1) after convolution with filter of size 3×3. The number of trainable parameters for this layer is 18,432 ((3*3*32)*64). Similarly, layer C_4 has 36,864 ((3*3*64)*64) trainable parameters and outputs 64 feature maps of size 6×6. No biases are used for second and third convolutional layers. The next two fully-connected layers FC_1 and FC_2 have 1024 and 512 nodes with 5,89,24 (576*1024) and 5,24,288 (1024*512) trainable parameters.

The last fully-connected layer, i.e. the output layer has 55 nodes and 28,215 ((512+1)*55) trainable parameters. Softmax function with cross-entropy loss function is used for the output layer. Batch normalization is adopted to combat the problem of internal covariate shift which is common in deep neural networks [89]. The last two fully-connected layers has a dropout layer of 0.3 between them to fight the problem of overfitting in neural network [204]. Mini-batch stochastic gradient descent optimization algorithm is used with a mini-batch size of 32 to train the proposed architecture. The network is trained in the same manner as the five CNN models considered earlier. That is, it is trained on the training and validation sets for 30 epochs with early stopping method with patience 10 and validation loss as the metric to be monitored.

Figure 3-10 provides a summary of the proposed CNN model. The total number of trainable parameters sums up to 1,207,351. The non-trainable 192 parameters come from the two batchnormalization layers used in the model. There is a total of 384 (128+256) parameters from the two batchnormalization layers out of which 192 (64+128) are non-trainable. These 192 non-trainable parameters are the computed mean and standard deviation of the activations which are not affected during training by backpropagation.

3.5 Experimental results and discussion

The performance of the proposed CNN is analysed in comparison with the different feature-classifier combinations discussed in Chapter 2 and the five pre-trained CNN models discussed in the present chapter. Performance analysis of the proposed CNN is also carried out by introducing some noise in the samples of TUMMHCD.

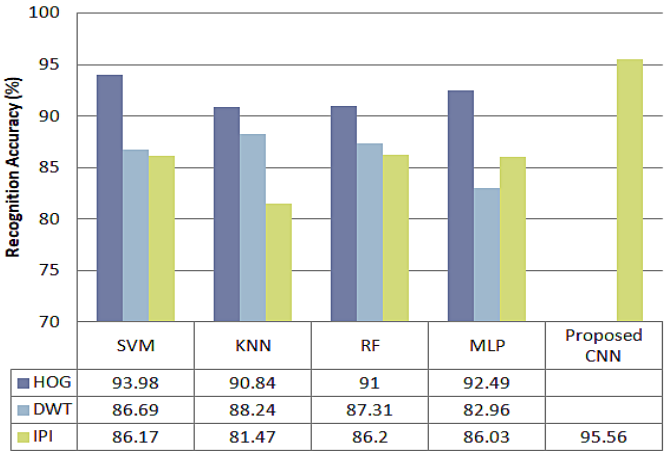
Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 22, 22, 32)	320
activation_8 (Activation)	(None, 22, 22, 32)	0
conv2d_7 (Conv2D)	(None, 20, 20, 32)	9216
activation_9 (Activation)	(None, 20, 20, 32)	0
batch_normalization_3 (Batch	(None, 20, 20, 32)	128
max_pooling2d_3 (MaxPooling2	(None, 10, 10, 32)	0
conv2d_8 (Conv2D)	(None, 8, 8, 64)	18432
activation_10 (Activation)	(None, 8, 8, 64)	0
batch_normalization_4 (Batch	(None, 8, 8, 64)	256
conv2d_9 (Conv2D)	(None, 6, 6, 64)	36864
activation_11 (Activation)	(None, 6, 6, 64)	0
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 64)	0
flatten_2 (Flatten)	(None, 576)	0
dense_4 (Dense)	(None, 1024)	589824
activation_12 (Activation)	(None, 1024)	0
dropout_2 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 512)	524288
activation_13 (Activation)	(None, 512)	0
dense_6 (Dense)	(None, 55)	28215
activation_14 (Activation)	(None, 55)	0

Total params: 1,207,543
Trainable params: 1,207,351
Non-trainable params: 192

Figure 3-10: Summary of details of each layer in the proposed CNN model

3.5.1 Analysis w.r.t feature-classifier combinations

Experimental results show that except for CNN, other classifiers perform poorly when fed with IPI values than with HOG and DWT features. However, when a deep network is employed a high test accuracy of 94.36% could be achieved with a base CNN. With the proposed CNN model, a test accuracy of 95.56% is achieved which sets the benchmark on the developed database. Test accuracy of the proposed CNN model is also tested on the famous MNIST dataset and an accuracy of 99.49% is achieved. Figure 3-11 provides the highest recognition accuracies achieved by different feature-classifier combinations. It can be observed from the experimental results that deep network outperforms other popular classifiers even without adopting a feature extraction technique. This shows the superiority of deep networks over other classification methods for the concerned dataset.



	SVM	KNN	RF	MLP	Proposed CNN
HOG	93.98	90.84	91	92.49	
DWT	86.69	88.24	87.31	82.96	
IPI	86.17	81.47	86.2	86.03	95.56

Figure 3-11: Recognition accuracies using different feature-classifier combinations

3.5.2 Analysis w.r.t pre-trained CNN models

A classwise classification rate of the five state-of-the-art CNN models and the proposed CNN model is given in Figure 3-12. Looking at the figure, one can observe that it shows a similar pattern as the ones shown by feature-classifier combinations in Chapter 2 (Figure 2-8). This indicates that even though the recognition accuracy is enhanced, classification rates of certain characters are still low when deep learning models are used for recognition.

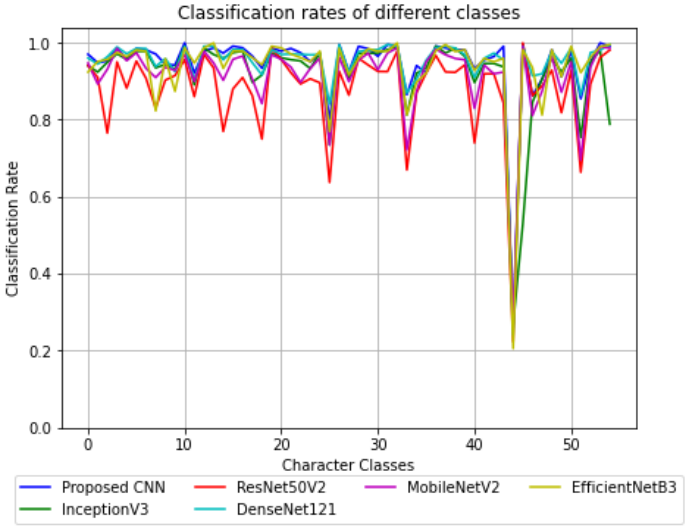


Figure 3-12: Classwise classification rates achieved using state-of-the-art CNN models and proposed CNN model.

A comparison in terms of different parameters of the proposed CNN model and the five CNN models is given in Figure 3-13. The proposed model outperforms the other state-of-the-art models. It also indicates that performance of CNN mod-

els depends on the dataset at hand and that deeper and more resource-demanding models are not necessarily the best option for certain datasets like TUMMHCD. Moreover, it is easier to carry out hyperparameter tuning for models with fewer layers as it takes less time to train and validate them.

CNN model	Input image size	Training time/epoch(s)		Validation accuracy		Total no. of layers in base model	Layer fine tuning starts	Test accuracy		Testing time(s)	
		TL	FT	TL	FT			TL	FT	TL	FT
InceptionV3	75x75	3844	3825	91.9	91.90	311	250	92.22	92.56	83.27	92.77
ResNet50V2	32x32	3768	4134	83.9	88.49	190	140	84.13	88.49	82.68	82.77
DenseNet121	32x32	994	826	16.6	91.15	427	370	17.18	90.81	83.98	81.96
MobileNetV2	32x32	447	465	79.9	93.51	154	100	79.47	93.49	10.75	10.26
EfficientNetB3	32x32	1523	1574	94.64	95.41	384	330	94.36	95.21	43.17	37.79
Proposed CNN	**24x24**	**162**		**95.6**		**20**	**-**	**95.56**		**6.57**	

Figure 3-13: Performance comparison of state-of-the-art CNN models and proposed CNN against TUMMHCD

3.5.3 Performance analysis of TUMMHCD with noise

In order to test the robustness of the proposed CNN model, experiments are conducted to test the accuracy of the model on TUMMHCD by introducing some noises in the samples of the dataset. Two types of noises are added viz. gaussian noise and salt and pepper noise. These noises are the most commonly found noises in images. For each type of noise, three variations of it are taken. Figure 3-14 shows the two types of noises and the three variations of each of them.

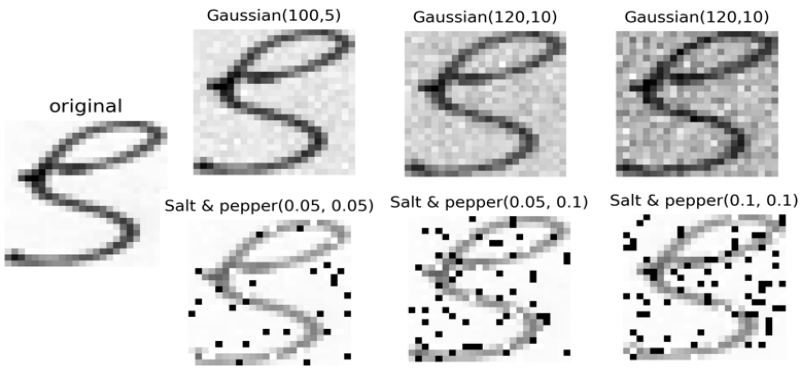


Figure 3-14: Different noises introduced in TUMMHCD. The values inside the brackets for the gaussian noise represent the mean and standard deviation respectively and those of salt and pepper noise indicate the percentages of salt and pepper noise in the image respectively.

Performance analysis is carried out by considering two scenarios: one where noise is added only to the test set and another where noise is added to

both the training and test sets. The noise is added to 5%, 10% and 15% of the samples, with equal percentages of each of gaussian and salt and pepper noises. For example, for 5% noise, 2.5% of gaussian (the three variations in equal amounts) and 2.5% of salt and pepper (the three variations in equal amounts) noises are introduced. The recognition results obtained are provided in Figure 3-15.

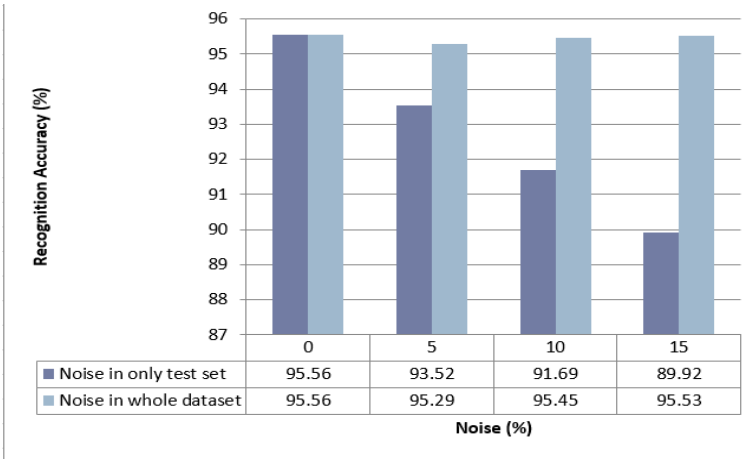


Figure 3-15: Recognition accuracies achieved when noises are added to TUMMHCD

When noises are introduced only in the test set, there is a decrease in the recognition accuracies of the proposed CNN. With noise in 5% of the samples in the test set, there is a decrease of 2.04% in the recognition accuracy. With 10% and 15% noisy samples, the recognition accuracies decrease by 3.87% and 5.64% respectively. This shows that although the proposed system could not classify all the noisy images correctly, it could classify more than half of the noisy images correctly. Moreover, the presence of more number of noisy images does not overwhelm the system as it becomes more robust in terms of the recognition accuracies achieved when more number of test images are noisy. The decrease in the recognition accuracy is expected as the system is trained on the original training set (without noise) and tested with the test set having noisy images. However, the proposed system could classify many of the noisy images correctly.

The second scenario is when the noises are introduced in the whole dataset, i.e. both the training and the test sets. This means that the CNN is trained and tested by adding noises to some of the samples. It is observed that there is an improvement in the recognition accuracies with more number of noisy samples. 5% noisy samples in both the training and test sets gives a recognition accuracy of 95.29%, 10% and 15% noisy images give 95.45% and 95.53% accuracies respectively. The addition of noisy images does not affect the performance of the system drastically as promising recognition accuracies are achieved with noisy images as

well. The proposed CNN can be made more robust with the introduction of more types of noises in the samples of the dataset.

3.6 Conclusion

This chapter presents a deep-learning based recognition of TUMMHCD. A CNN model is built from scratch which outperforms recent state-of-the-art pre-trained CNN models. Experimental results show the presence of certain characters which are misclassified a greater number of times by all the CNN models as well as by traditional feature-classification approach. Efforts are to be made to increase the classification rates of such characters so as to enhance the overall recognition accuracy of the system. In the two chapters that follow, approaches are proposed to achieve this task.