# Chapter 6

# Using LSTM language model with CNN for handwritten character recognition

## 6.1 Introduction

This chapter explores the combination of LSTM language model (LM) and CNNs for handwritten character recognition of Meitei Mayek. The primary goal is to leverage the strengths of both architectures: CNNs to extract visual features for image understanding and LSTMs for contextual understanding. The proposed methodology aims to improve the recognition accuracy by considering not only the visual information of characters but also the contextual dependencies between characters in a text sequence.

Convolutional Neural Networks (CNNs) have demonstrated remarkable success in image-based recognition tasks, including handwritten character recognition. CNNs excel at capturing the visual features from images. However, their ability to model contextual dependencies and understand the sequential nature of handwriting is limited. To overcome this limitation, researchers have been adopting the concept of language modelling in the HCR systems. Language models are well-suited for modelling sequential data and capturing long-range dependencies. When we talk about language models, they can be broadly divided into two categories, viz., statistical language models and neural network language models (NNLMs). Statistical models are based on traditional statistical methods like N-grams and Hidden Markov Models (HMM) to learn the probability distribution of characters and/or words. They play a significant role in many applications such as speech and character recognition, information retrieval and machine translation, etc. Many handwritten text recognition (HTR) technologies have been developed based on N-grams and HMMs [215, 216]. The HTR contests on the TRANSCRIPTORIUM[181] datasets see many entries with different methodologies using HMMs, N-gram LMs and neural networks [182, 183]. The more recent competitions which are performed on the READ dataset report on the use of CNN, RNN, Bidirectional LSTM (BLSTM) along with N-gram language models [184, 185]. The back-off N-gram language models (BLMs) have also been widely adopted in many text recognition systems [35, 57, 128, 227, 242]. The drawback with statistical models, however, is that they suffer from the curse of dimensionality and data sparseness problem. This means that in order to achieve a more accurate estimate of the sequence probability, the statistical models should be of higher order. Carpenter [36] pointed out that with sufficient training samples, the performance of an N-gram model can be improved until 8-gram. However, higher N leads to a lot of computation overhead as the number of parameters increases exponentially. Moreover, N-grams are a sparse representation of language. This is because the model is built based on the probability of words co-occurring. It will give zero probability to the words that are not present in the training corpus. Recently, the NNLM has been introduced to address the data sparseness problem [27]. Since then, NNLMs have been successfully applied to many machine learning tasks such as speech recognition [133, 191] and machine translation [110, 192] and handwriting recognition [232, 239].

By incorporating language models into recognition systems, accuracy can be enhanced, challenging scenarios can be handled, and the overall quality of recognized text can be improved, making it a valuable component of handwriting recognition technology. When we talk about incorporating LM in HCR systems, there can be two ways of doing it. One is when the LM is integrated into the HCR system and works as an end-to-end approach and the other is when LM is used as a post-processing step after the recognition where it helps bridge the gap between the raw character recognition output and the final, high-quality textual representation. The first technique requires a large image dataset consisting of pages of handwritten texts along with the ground truth or transcriptions in order to train the system. This type of dataset is currently not available for the concerned script. The creation of such datasets will be taken up as part of our future work.

In the present work, NNLM is used as a post-processing step but at the same time, the decision that the recognition step makes, CNN in the present case, is also taken into account in delivering the final decision of the system. Therefore, the proposed methodology is able to capture both the class probabilities of the CNN based on the visual features and the conditional probabilities of the NNLM based on the contextual information. Both the language context model and visual model are found to be of great importance in handwritten text recognition [227]. The two types of probabilities are then combined to generate the final probability. Weighted sum approach is used for the combination of the two types of probabilities.

## 6.2 Related work

In the context of HTR of historical documents, combination of optical models and language models has been extensively explored. The use of optical models such as HMMs and the classic N-gram models have been found in many works reported in literature. In the work reported by Sanchez et al. [182], the best performance on the TRANSCRIPTORIUM datasets (as of 2014) are obtained by adopting Multi-directional Long Short-Term Memory (MDLSTM) NN as optical model and a lexicon derived from the training transcripts which performs as a 1-gram LM. A similar methodology achieves the best results in the HTR ICDAR-2015 contest as well [183]. For the READ datasets, the best performance in the HTR ICFHR-2016 is shown by CNN having five layers followed by MDLSTMs as the optical model and a character 10-gram LM [184]. The HTR ICDAR-2017 contest [185] reports the best performance by adopting a seven-layered CNN and two-layered of BLSTM, along with a character 10-gram for decoding. The work presented by Sanchez et al. [186] summarizes the results of the four contests viz. ICFHR-2014, ICDAR-2015, ICFHR-2016 and ICDAR-2017. The work also reports on the benchmark HTR technologies achieved against the datasets. They could achieve better results by the adoption of plain BLSTM architecture instead of a MDLSTM and a higher-order character N-gram models such as character 7-gram or 8-gram. More number of training data are also generated to train the optical models as part of their work.

Marti and Bunke [128] propose an enhanced HMM-based handwriting recognition system using a statistical LM. The approach avoids the need to segment lines of text into individual words. The work of Wang et al. [226] presents an approach of integrating segmentation and recognition steps for offline handwritten Chinese text recognition. Four types of character-level and word-level LMs are evaluated and it is found that character-trigram (charTri) performs the best for character recognition and segmentation out of charBi, charTri, wordBi and wordTri. Kang et al. [92] propose a method of integrating an external language model with handwritten word recognizer in which it has the ability to choose or discard the information provided by the language model. They achieve state-of-the-art results on IAM, GW and Rimes datasets. The work of Neto et all. [141] presents spelling correction techniques for text-processing which eliminates the linguistic dependence between the optical model and the decoder. They also propose a training methodology for NN encoder-decoder architecture for the purpose of spelling correction. Experiments on five well known datasets provide state-of-the-art performance.

In the work of Wick et al. [229], a bidirectional transformer-based encoder-decoder is proposed which performs decoding in reading-order and also reverse directions. They also pointed out the need for a large training dataset for transformers. With a larger dataset, they are able to outperform their reference model by 26%. In a simple experimented conducted by Gold and Zesch [64], they are able to enhance the HTR system by incorporating a unigram language model as a post-processing step to the recognizer which consists of five CNN layers, two RNN layers and a Connectionist Temporal Classification (CTC) layer. The recognizer is trained on IAM dataset. Hu et al. [78] employ a two-pass recognition approach. The first pass uses a common LM to retrieve the relevant contents from the internet by obtaining the initial recognition results. Then an adaptive LM is generated based on the related contents. The two LMs are then combined for the second-pass recognition. Significant performance improvements are seen in CASIA-HWDB and ICDAR-2013 datasets. Another work [203] proposes a method for adapting a generic HTR system to a specific writer. The optical model composes of eight CNN layers followed by two BLSTM layers. The decoding layer is a 9-gram LM of 2-multigrams. The approach is evaluated on the READ dataset and reports comparable performance with state-of-the-art results.

In a very recent work of Li et al. [115], pre-trained transformer and text transformer models are used to leverage the transformer architecture for both image understanding and text generation. They show that the proposed approach outperforms state-of-the-art models on handwritten, printed and scene text recognition tasks. Another recent work based on transformer [134] introduces two architectures viz. Transformer Transducer and the standard sequence-to-sequence Transformer. Pre-trained transformers are employed as both optical and language models. They achieve state-of-the-art performance on the Arabic KHATT dataset. The approach achieves a more parallelizable and less complex system by modelling language dependencies and focusing on the attention mechanism.

From the works reported in literature, it is found that existing typical HTR systems utilize combinations of CNNs, RNNs and CTC for text generation and an explicit language model as a post-processing step to improve the overall accuracy of the system. The latest HTR technology utilizing transformer requires a significantly large labelled dataset of handwritten or printed texts. In Meitei Mayek, there is not a single such dataset available currently, which can be used to even fine-tune hyperparameters if one wishes to use pre-trained models also. Therefore, LSTM-based LM is utilized in the present work which requires comparatively smaller dataset to train and it has been trained using a text corpus. The work described in this chapter is an attempt to evaluate the performance of a CNN-based HCR system for Meitei Mayek with the incorporation of a LM.

## 6.3 Proposed methodology

The conceptual framework of the proposed methodology is shown in Figure 6-1. The training of CNN is carried out using the TUMMHCD dataset as described in Section 4.3.1. The LSTM-based language model is trained at character-level using the TDIL text corpora named *The TDIL Hindi-Manipuri Agriculture & Entertainment Text Corpus ILCI-II*[1]. The details of the corpus have already been provided in Chapter 5, under Section 5.3. For the evaluation of the proposed method, the test dataset used is the 100 words dataset described in Section 5.5.1. Pre-processing takes place on the test images following the steps described in Section 5.4.3 and Figure 5-11. After pre-processing, the segmented test character images are fed to the trained CNN which gives out the raw output character text. The CNN output text is given as the input to the trained LSTM language model (LSTM-LM). The proposed approach then combines the class probabilities and conditional probabilities of the characters in each of the test word image to give the final output text. Different steps of the proposed strategy are described in the following sections.

[1]https://tdil-dc.in/index.php?option=com_download&task=showresourceDetails&toolid=1874&lang=en

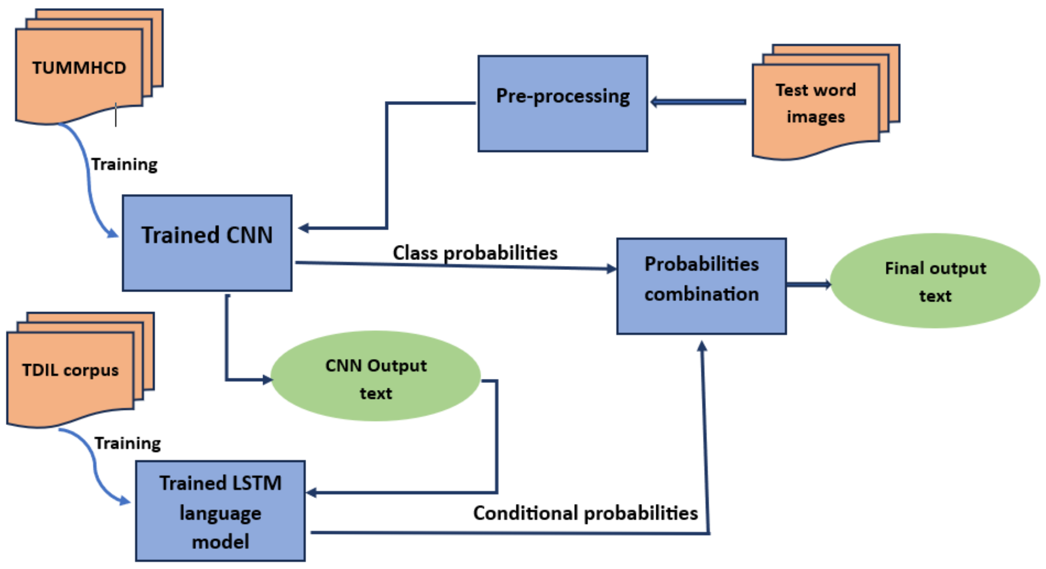


**Figure 6-1:** Conceptual framework of the proposed CNN-LSTM recognition system

### 6.3.1 CNN training

The training of CNN is carried out in the same manner as described in Section 4.3.1. The difference here is with the TUMMHCD dataset that has been used. The model is trained with only 54 character classes (excluding the character class representing ”।।”), since it does not represent a word. The reason why this change is considered will become more clear with the subsequent sections.

### 6.3.2 LSTM language model

The character-level LSTM language model is trained using the TDIL corpus. For the purpose of training, the corpus is considered word by word, which is described in the subsections that follow. Data cleaning is carried out on the original corpus in order to make it appropriate for the purpose.

#### 6.3.2.1 Data cleaning

Firstly, certain characters such as *(, ), /,* -, ., etc. are removed. The character ”।।” which signifies the end-of-line is also removed from the original corpus since it does not represent a word. The inclusion of ”।।” is more relevant in those scenarios where the model is to be trained for word-level language modelling, and not character-level modelling like the present work. Thus, the cleaned text corpus has 55 characters, 54 Meitei Mayek characters (excluding "||") and the whitespace.

#### 6.3.2.2 Training LSTM language model

For training the LSTM, the input sequences are generated by considering the text corpus at word level. That is, the sequences are generated for each of the words separately. For example, if the first word $W_1$ has the character sequence $< ch_1, ch_2, ch_3, ch_4, ch_5 >$, the generated sequences with respect to this particular word are: $[0, 0, 0, ..., 0, t_1, t_2]$, $[0, ..., 0, 0, t_1, t_2, t_3]$, $[0, ..., 0, t_1, t_2, t_3, t_4]$ and $[0, .., t_1, t_2, t_3, t_4, t_5]$, where $t_i$ is the token for character $ch_i$ and 0's are added to give a fixed sequence length. Similarly, the second word generates its corresponding set of sequences, the third word, its corresponding set of sequences and so on and so forth. The sequence length used is 22 which is the length of the longest word present in the text corpus. A total of 8,04,292 sequences are generated to train the LSTM. The vocabulary size is 55 and therefore there are 55 tokens, each token representing a character.

The LSTM LM is trained using the text corpus such that given a sequence of characters, it is able to predict the most likely character to come next. It works in a similar way like the popular word2vec models. Once the LSTM is trained, the trained model is used to generate the conditional probabilities of the input sequence of tokens (characters in this case). Since the present work deals with tokens that represent the characters, the term "token" would mean token representing characters in the rest of the thesis. The input to the LSTM is the output text given out by the CNN. The LSTM-LM estimates the probability distribution of the next token in the sequence, known as conditional probability based on the prior context and uses this distribution to predict the next token. The prior probability of the entire sequence is the product of the token conditional probabilities along the sequence. The model is trained to minimize the difference between these predicted probabilities and the ground truth labels for each token.

Let $X = (x_1, x_2, x_3, ..., x_t)$ be the input sequence to the LSTM LM, where $x_t$ represents the current token in the sequence, and $t$ is the position in the sequence. The LSTM model maintains a hidden state $h_t$ at each position $t$, which represents the context learned from the preceding characters in the sequence. At each time step $t$, the LSTM updates the hidden state $h_t$ based on the current input $x_t$ and the previous hidden state $h_{t-1}$. This update is defined by LSTM equations, which include operations like input gates, forget gates, and output gates:

$$h_t = LSTM(x_t, h_{t-1}) \tag{6.1}$$

After updating the hidden state $h_t$, the LSTM passes it through the output layer which generates a probability distribution over the vocabulary of possible tokens. Let $P_t$ represent this distribution at time step $t$.

$$P_t = OutputLayer(h_t) \tag{6.2}$$

$P_t$ is the conditional probability distribution over all possible tokens, and each element $P_t[i]$ represents the probability of the $i^{th}$ token in the vocabulary being the next token in the sequence. During training, the model is provided with the ground truth labels for the next token in the sequence, denoted as $y_t$. The training objective is to minimize the categorical cross-entropy loss between the predicted distribution $P_t$ and the actual target distribution $y_t$.

The total 8,04,292 sequences are divided randomly into training set and validation set in the ratio 9:1. The training set has 7,23,862 sequences and test set has 80,430 sequences. The training is carried out in the cloud-based Jupyter notebooks provided by Kaggle. A baseline model having one LSTM layer and 100 hidden units is first taken for training. A dropout layer of 0.5 is used after the LSTM layer followed by a dense layer with softmax activation function. The optimizer used is Adam. The training is done for 100 epochs with early stopping method with a patience of 20 on the validation loss. The batchsize taken is 32. The baseline model achieves an accuracy of 56.20% on the validation set and 58.33% on the training set. The number of hidden units is increased to 700 and a dropout layer of 0.9 is used. The batch size is updated to 1024 for a faster training, keeping the other parameters same. With this architecture, the LSTM LM reaches a validation accuracy of 65.20% and a training accuracy of 68.40%. After this point, the model could not perform better with the addition of layers or hidden units or with changing the batchsize. Once the model is trained, it is used to generate the conditional probabilities of the test input sequence. The input sequence is the sequence of tokens representing the character text generated by the CNN.

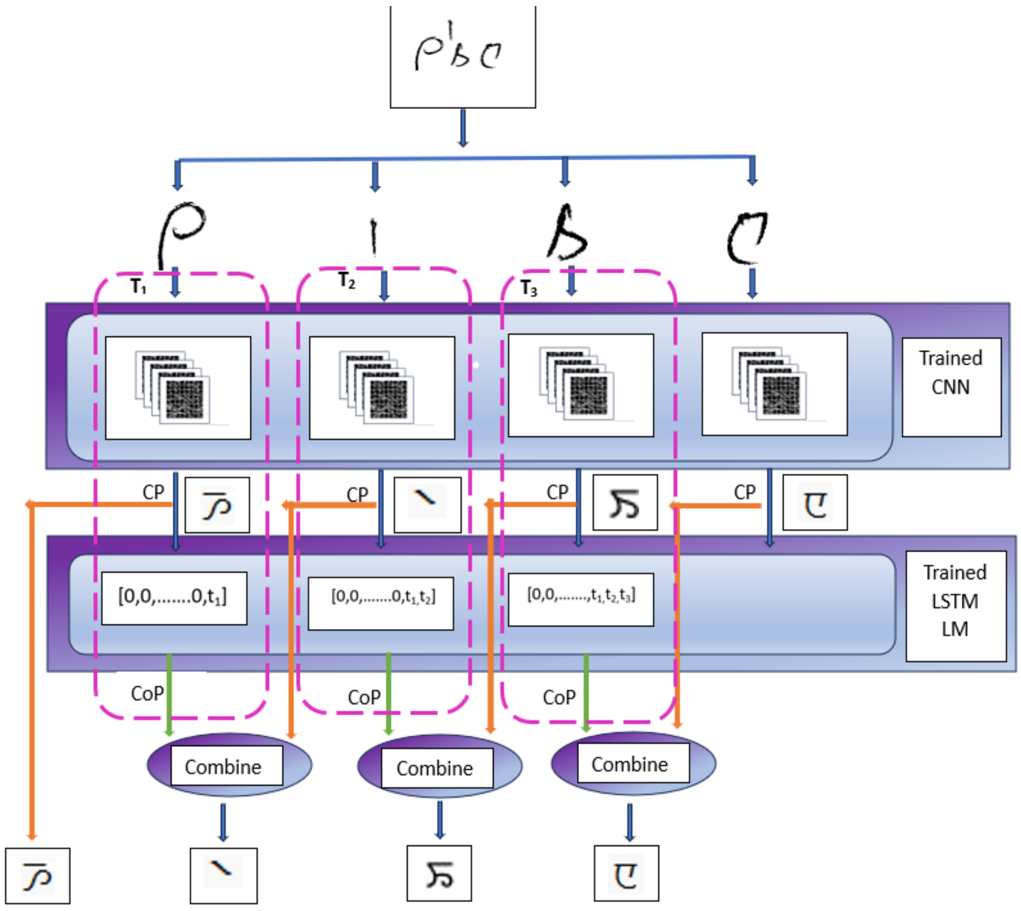
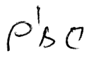


**Figure 6-2:** The workflow of the test images through the trained CNN and trained LSTM-LM. The orange lines depict the class probabilities (CP) generated by the CNN and the green lines depict the conditional probabilities (CoP) generated by the LSTM-LM.

### 6.3.3 Testing of test images

The workflow of the test images through the trained CNN and LSTM models is shown in Figure 6-2. The input test word image  is first segmented into its constituent characters. In time step $T_1$, the first character " " is fed to the trained CNN which extracts the visual features and based on these features gives the CPs. The class of the final corresponding output text is the one with the highest CP. In the same time step, the input sequence to the trained LSTM-LM is generated. The generated input sequence is $[0, 0, ...., t_1]$ where $t_1$ is the token of the first character " ". Based on this input sequence, the LSTM generates the CoPs which is a probability distribution of the next token over the vocabulary.

In time step $T_2$, the second character " " is fed to the CNN and the corresponding CPs are generated based on the visual features. The idea is to leverage both the visual information and contextual information. Therefore, the

CoPs generated in the previous time step $T_1$ is combined with the CPs generated in time step $T_2$ to give the final output text corresponding to the second character ” ”. The input sequence given to the LSTM at time step $T_2$ is $[0, 0, ...., t_1, t_2]$, where $t_1$ and $t_2$ are the tokens corresponding to the first and second characters respectively. This input sequence generates the CoPs at time step $T_2$. At time step $T_3$, the third character ” ” enters the CNN. The CoPs which are generated in time step $T_2$ are combined with the CPs generated in time step $T_3$ to give the final output text corresponding to the third character ” ”. Similarly, at each time step $T_i$, the $i^{th}$ character is fed to the CNN and the CPs generated at this time step is combined with the CoPs generated in the time step $T_{i-1}$ to produce the final output text corresponding the the $i^{th}$ character. This is continued till the last character in the word is reached.

**Probabilities combination:** The CoPs generated by the LM and the CPs generated by the CNN are combined for each of the characters in a word in the test dataset, except for the first character. For the first character in each of the words, the final output text is based only on the class probability which is the output of the CNN. The character will be classified to the class with the highest class probability. From the second character onwards, the CPs of the character along with the CoPs are considered to generate the final output text (Refer Figure 6-2). For combining the probabilities from the two different models, a weighted sum approach is employed. The order of CoPs in the distribution vector corresponds to the order of tokens in the vocabulary. The probability at each index $i$ in the vector represents the likelihood of the token at index $i$ in the vocabulary being the next token in the sequence. Therefore, the CNN is also trained in such a way that the generated vector of CPs also follows the same order as that of CoPs by assigning proper class labels to the character classes. The distribution vector size of the CPs and the CoPs is however the same as generated CoPs do not include CoP for the token representing the whitespace.

For an input character image, let the CP vector is represented by:
$< a_1, a_2, ..., a_{54} >$,
where $a_i$ is the probability of the character belonging to class $i$
And, let the CoPs vector is represented by:
$< b_1, b_2, ..., b_{54} >$
where $b_i$ is the probability of the character belonging to class $i$
Then, the final probability vector is given by:
$< p_1, p_2, ..., p_{54} >$,
where $p_i = w_c * a_i + w_l * b_i$, where $w_c$ and $w_l$ are the weights given to the CPs and

Table 6.1: Results obtained with different weights

| Weights CP | CoP | Accuracy |
|---|---|---|
| 0.5 | 0.5 | 90.33% |
| 0.3 | 0.7 | 87.20% |
| 0.7 | 0.3 | 92.92% |

CoPs respectively. The character finally gets classified to the class $i$ corresponding to the highest $p_i$. Experiments have been carried out by giving different weights to the two types of probabilities. The results are discussed in Section 6.4.

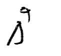
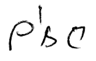
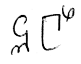
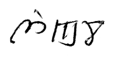
## 6.4 Experimental results

It is observed that the accuracy achieved by the LSTM-LM, as expected, is not as good as that of the CNN. Therefore, it is necessary to also understand that while combining the two types of probabilities, the weights should be given appropriately. In order to have a better understanding of the two models, experiments are carried out to calculate the weighted sum by considering three different weights. The class with the highest weighted sum of the two probabilities is the final class of the character. The results are provided in Table 6.1.

The accuracies vary moderately with the change in the weights assigned to the two types of probabilities. As a baseline, equal weights of 0.5 are assigned to the CNN and to the LSTM. With this weight assignment, the recognition accuracy achieved is 90.33% which is slightly better than what is achieved with only CNN. When the weights assigned are changed to 0.3 for the CNN and 0.7 for the LSTM, there is a reduction in the recognition accuracy (87.20%). This means that with the increased weightage given to the LSTM-LM, the system performs worse. This is either because the characters which are misclassified by the CNN are not rectified even after combining the CPs and CoPs, or the characters which are classified correctly by the CNN gets misclassified after the combination of the two probabilities. The proposed approach with weights of 0.7 and 0.3 assigned to the CNN and the LSTM-LM, respectively achieves the maximum recognition accuracy of 92.92%, f1 measure of 93.32%, precision of 94.41% and recall of 92.92%.

Some of the test samples with the corresponding CNN-classified and LM-classified characters are given in Table 6.2. In the first sample, based on the first character generated by the CNN (ꯁ), the LM generates a wrong character ꯎ.

Table 6.2: Test samples with classification results of CNN, LSTM-LM and CNN+LSTM-LM. The last column provides the correct sequence of characters.

| Sample | CNN | LSTM-LM | CNN+LSTM Output | Correct output |
|---|---|---|---|---|
|  | ꯅꯧ | ꯕ | ꯅꯧ | ꯅꯧ |
|  | ꯄꯣꯅꯂ | ◌ꯣ ꯅꯕ | ꯄꯣꯅꯂ | ꯄꯣꯅꯂ |
|  | ꯆꯂꯏ | ꯄꯤ | ꯆꯂꯤ | ꯆꯂꯤ |
|  | ◌ꯨ◌ꯣꯡꯕ | ꯐꯔ◌ꯤ | ◌ꯨ◌ꯣꯡꯕ | ꯃꯣꯡꯕ |

However, the combined system (CNN+LSTM) is able to rectify such errors which finally outputs the correct sequence of characters. For the second sample, based on the first character ꯄ, the LSTM is able to generate the next character correctly (ꯄ) and also able to generate the third character correctly (ꯅ) based on the previous two characters (ꯄꯣ). Note that the third character is not recognised correctly (ꯅ) by the CNN. Combining the two models, the system is able to produce all the four characters correctly. This is because of the low confidence with which the CNN predicts the third character based on the visual features whereas the LM predicts it with high confidence based on the two previous characters. For the third sample, the CNN predicts the first two characters correctly (ꯆꯂ) while it fails to predict the third one correctly (ꯏ). The LSTM-LM, on the other hand, predicts the first character wrongly (ꯄ) based on the first character (ꯆ). However, it predicts the third character correctly (◌ꯤ) based on the previous two characters (ꯆꯄ). With the last test sample in the table, the CNN predicts the first character wrongly (◌ꯨ) and the last three are predicted correctly (◌ꯣꯡꯕ). Based on the wrongly predicted first character, the LSTM-LM produces all the three succeeding characters wrongly (ꯐꯔ◌ꯤ). The final output sequence of characters is however close to the correct output with only the first character misclassified. From the results shown in Table 6.2, it is observed that the proposed system is performing better by giving more weightage to the output generated by the CNN. What it is exactly doing is in those cases where the CNN classifies the characters with less confidence, the LSTM-LM having more confidence plays a role in delivering the correct final sequence of characters based on the contextual information. The proposed methodology of incorporating LM with CNN therefore enhances the overall recognition accuracy of the system. The confusion matrix obtained by the proposed CNN+LSTM system is shown in Figure 6-3. Table 6.3 lists the precision, recall and f-measure of each class obtained with the proposed system.

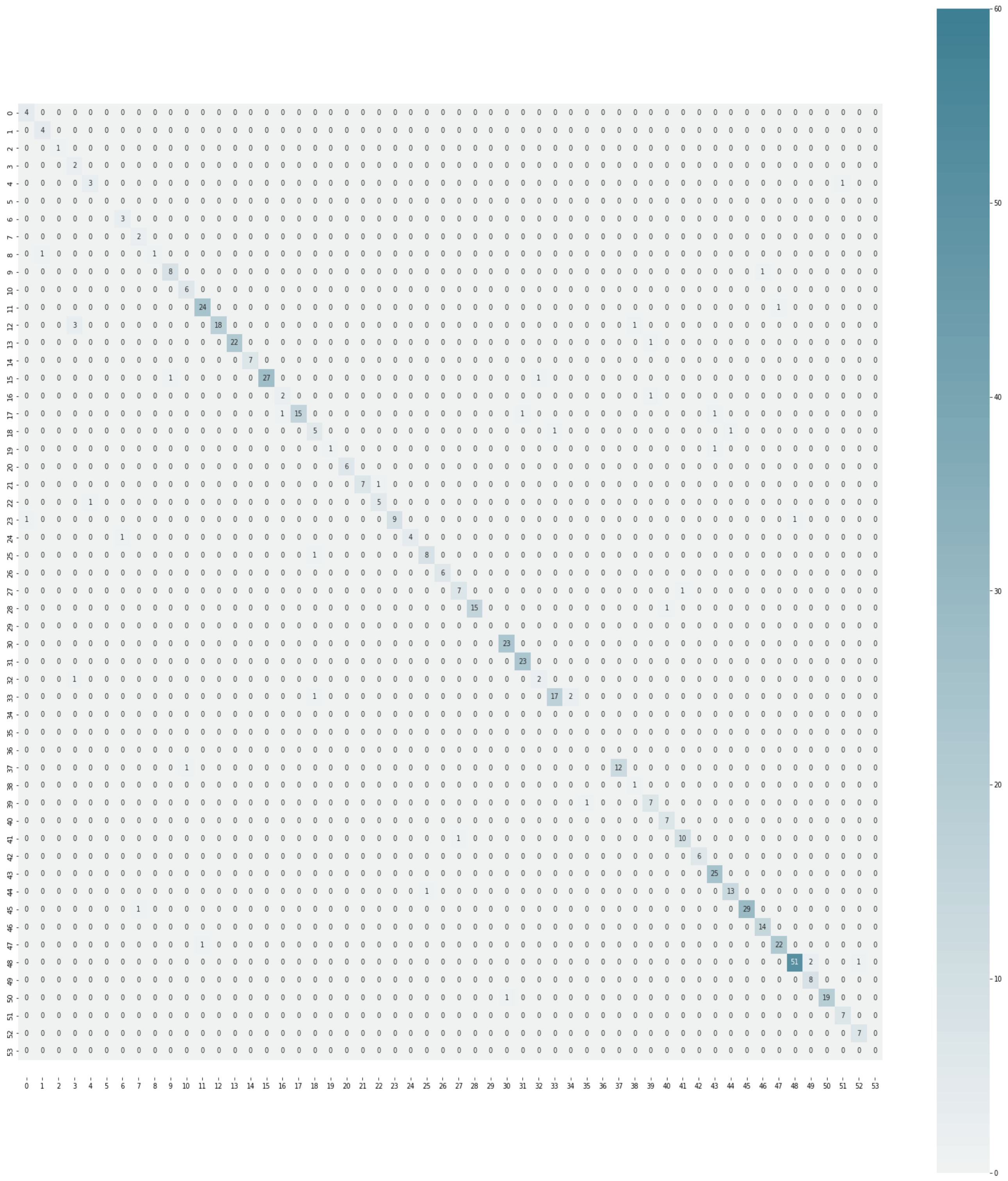


**Figure 6-3:** Confusion matrix obtained using the proposed CNN+LSTM system

Table 6.3: Total number of instances and values of precision, recall and f-measure of each class obtained with systems with and without the incorporation of LM. The highlighted values in bold indicate the improvements.

| Character class (Symbol) | Number of instances | Without LM Precision | Without LM Recall | Without LM F-measure | With LM Precision | With LM Recall | With LM F-measure |
|---|---|---|---|---|---|---|---|
| 0 (ϩ) | 4 | 0.75 | 0.75 | 0.75 | 0.80 | 1.00 | 0.89 |
| 1 (ꝗ) | 4 | 0.80 | 1.00 | 0.89 | 0.80 | 1.00 | 0.89 |
| 2 (ꝙ) | 1 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 3 (ȣ) | 2 | 0.33 | 1.00 | 0.50 | 0.33 | 1.00 | 0.50 |
| 4 (ꝓ) | 4 | 0.75 | 0.75 | 0.75 | 0.75 | 0.75 | 0.75 |
| 5 (ꝑ) | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 (ℋ) | 3 | 0.75 | 1.00 | 0.86 | 0.75 | 1.00 | 0.86 |
| **7** (ꝕ) | **2** | **0.33** | **0.50** | **0.40** | **0.67** | **1.00** | **0.80** |
| 8 (ꝑ) | 2 | 1.00 | 0.50 | 0.67 | 1.00 | 0.50 | 0.67 |
| **9** (O) | **9** | **0.73** | **0.89** | **0.80** | **0.89** | **0.89** | **0.89** |
| 10 (⊞) | 6 | 0.86 | 1.00 | 0.92 | 0.86 | 1.00 | 0.92 |
| **11** (ო) | **25** | **0.85** | **0.88** | **0.86** | **0.96** | **0.96** | **0.96** |
| 12 (Շ) | 22 | 1.00 | 0.82 | 0.90 | 1.00 | 0.82 | 0.90 |
| 13 (Ѭ) | 23 | 1.00 | 0.96 | 0.98 | 1.00 | 0.96 | 0.98 |
| 14 (ɱ) | 7 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 15 (Ꮳ) | 29 | 1.00 | 0.93 | 0.96 | 1.00 | 0.93 | 0.96 |
| 16 (Ꭱ) | 3 | 0.67 | 0.67 | 0.67 | 0.67 | 0.67 | 0.67 |
| 17 (Ꮫ) | 18 | 0.94 | 0.83 | 0.88 | 1.00 | 0.83 | 0.91 |
| 18 (ɲ) | 7 | 0.71 | 0.71 | 0.71 | 0.71 | 0.71 | 0.71 |
| 19 (Ш) | 2 | 1.00 | 0.50 | 0.67 | 1.00 | 0.50 | 0.67 |
| 20 (ꟻ) | 6 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 21 (Ꮘ) | 8 | 1.00 | 0.88 | 0.93 | 1.00 | 0.88 | 0.93 |
| 22 (Ꭱ) | 6 | 0.83 | 0.83 | 0.83 | 0.83 | 0.83 | 0.83 |
| 23 (ᓄ) | 11 | 1.00 | 0.82 | 0.90 | 1.00 | 0.82 | 0.90 |
| 24 (ꝙ) | 5 | 1.00 | 0.80 | 0.89 | 1.00 | 0.80 | 0.89 |
| **25** (Ꮥ) | **9** | **0.29** | **0.44** | **0.35** | **0.89** | **0.89** | **0.89** |
| 26 (Ɛ) | 6 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 27 (Ɯ) | 8 | 0.88 | 0.88 | 0.88 | 0.88 | 0.88 | 0.88 |
| 28 (Ⲡ) | 16 | 1.00 | 0.94 | 0.97 | 1.00 | 0.94 | 0.97 |
| 29 (Ɛ) | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 30 (Ɐ) | 23 | 0.96 | 1.00 | 0.98 | 0.96 | 1.00 | 0.98 |
| 31 (Ɣ) | 23 | 0.96 | 1.00 | 0.98 | 0.96 | 1.00 | 0.98 |
| 32 (Շ) | 3 | 0.67 | 0.67 | 0.67 | 0.67 | 0.67 | 0.67 |
| 33 (ꝺ) | 20 | 0.94 | 0.80 | 0.86 | 0.94 | 0.85 | 0.89 |
| 34 (Ѫ) | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 35 (J) | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 36 (ᒣ) | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 37 (Ш) | 13 | 1.00 | 0.92 | 0.96 | 1.00 | 0.92 | 0.96 |
| 38 (ᒣ) | 1 | 0.50 | 1.00 | 0.67 | 0.50 | 1.00 | 0.67 |
| 39 (Ħ) | 8 | 0.78 | 0.88 | 0.82 | 0.78 | 0.88 | 0.82 |
| 40 (Ɱ) | 7 | 0.88 | 1.00 | 0.93 | 0.88 | 1.00 | 0.93 |
| 41 (Ɐ) | 11 | 0.91 | 0.91 | 0.91 | 0.91 | 0.91 | 0.91 |
| 42 (Ⴑ) | 6 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 43 (Ш) | 25 | 0.93 | 1.00 | 0.96 | 0.93 | 1.00 | 0.96 |
| **44** (Ꮥ) | **14** | **0.62** | **0.36** | **0.45** | **0.93** | **0.93** | **0.93** |
| 45 (◌̀) | 30 | 1.00 | 0.97 | 0.98 | 1.00 | 0.97 | 0.98 |
| **46** (◌°) | **14** | **0.85** | **0.79** | **0.81** | **0.93** | **1.00** | **0.97** |
| **47** (◌̥) | **23** | **0.86** | **0.83** | **0.84** | **0.96** | **0.96** | **0.96** |
| 48 (◌ſ) | 54 | 0.98 | 0.94 | 0.96 | 0.98 | 0.94 | 0.96 |
| 49 (◌̉) | 8 | 0.78 | 0.88 | 0.82 | 0.80 | 1.00 | 0.89 |
| 50 (◌̛) | 20 | 1.00 | 0.95 | 0.97 | 1.00 | 0.95 | 0.97 |
| **51** (◌̉) | **7** | **0.60** | **0.86** | **0.71** | **0.88** | **1.00** | **0.93** |
| 52 (◌̉) | 7 | 0.88 | 1.00 | 0.93 | 0.88 | 1.00 | 0.93 |
| 53 (‖) | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 54 (␣) | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

## 6.5 Conclusion

This chapter introduces the incorporation of an LSTM-based LM with CNN for the recognition of handwritten Meitei Mayek characters. It works by leveraging the power of CNN which is based on the visual representation and that of the LM which is based on contextual information. The proposed system employs combination of the two models by assigning more weightage to the CNN and achieves a better recognition accuracy on the handwritten Meitei Mayek characters. The proposed system overcomes some of the limitations of the methodology presented in Chapter 5 viz. script-specific and orthogonal properties which cannot be generalized and zone identification which may be prone to errors. The work, however still considers words with non-touching characters. Through this work, it is intented to carry forward the study of unconstrained handwriting of Meitei Mayek recognition by developing appropriate datasets.