Problem Statement

The task is a multi-classification problem focused on predicting the administrative **state/region** from a given **village name**. Village names are often short and unique, making this a challenging task. High precision and recall are critical, as the model's predictions will be used in downstream processes where geographic accuracy is essential. Effective encoding and feature representation of names are key factors in model performance.

Dataset Exploration

During initial exploration, it was observed that **some village names appear in multiple states or regions**, indicating **name duplication across different administrative divisions**. This introduces **label ambiguity**, which can negatively affect model performance by confusing the classifier during training. To address **data quality issues**, common noisy characters such as parentheses () and slashes / were removed using **regular expressions**. This preprocessing step ensures that the input names are clean and standardized before being passed to the tokenization pipeline.

Furthermore, an analysis of the class distribution revealed a **significant imbalance among the 18 target classes (states/regions)**. As illustrated in **Figure 2**, some regions are heavily overrepresented, while others have relatively few examples. This imbalance may lead to biased model predictions favouring the majority classes.

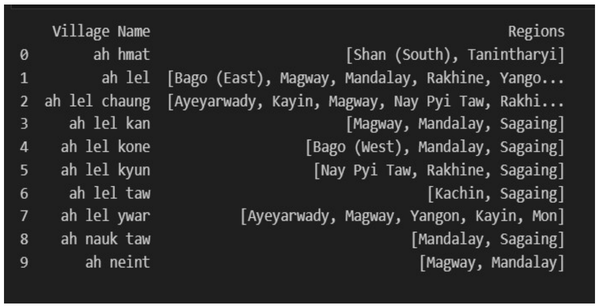
Considering Phonetic Normalization for Romanized Burmese Names

To improve classification accuracy for Romanized Burmese village names, a **phonetic normalization technique** was applied to address the variations in pronunciation across different ethnic groups. Burmese place names are often transcribed in diverse ways due to regional and ethnic dialects, leading to inconsistent spelling patterns that can hinder model learning.

This normalization process systematically standardizes common phonetic variants by:

- **Reducing aspirated and compound consonants** (e.g., ph, hp, hs) to their base forms (p, s).
- **Unifying vowel and diphthong representations**, such as converting au, aw, or ay to o or e.
- **Harmonizing tone variations and endings** (e.g., aung → ong, ein → en).
- **Removing redundant characters** like double consonants (pp → p, ll → l).
- **Lowercasing, trimming, and cleaning** the input to retain only alphabetic characters.

By mapping these phonetic differences into consistent forms, the model can better learn meaningful patterns and generalize across dialect-influenced spelling differences. This preprocessing step aims to reduce noise in the input and enhance the classifier's ability to focus on true linguistic signals.



```
    Village Name                                        Regions
0        ah hmat                  [Shan (South), Tanintharyi]
1         ah lel  [Bago (East), Magway, Mandalay, Rakhine, Yango...
2  ah lel chaung  [Ayeyarwady, Kayin, Magway, Nay Pyi Taw, Rakhi...
3     ah lel kan                  [Magway, Mandalay, Sagaing]
4    ah lel kone             [Bago (West), Mandalay, Sagaing]
5    ah lel kyun            [Nay Pyi Taw, Rakhine, Sagaing]
6     ah lel taw                          [Kachin, Sagaing]
7    ah lel ywar       [Ayeyarwady, Magway, Yangon, Kayin, Mon]
8    ah nauk taw                        [Mandalay, Sagaing]
9       ah neint                         [Magway, Mandalay]
```

Fig 1: Village Names with Multi-Region Presence

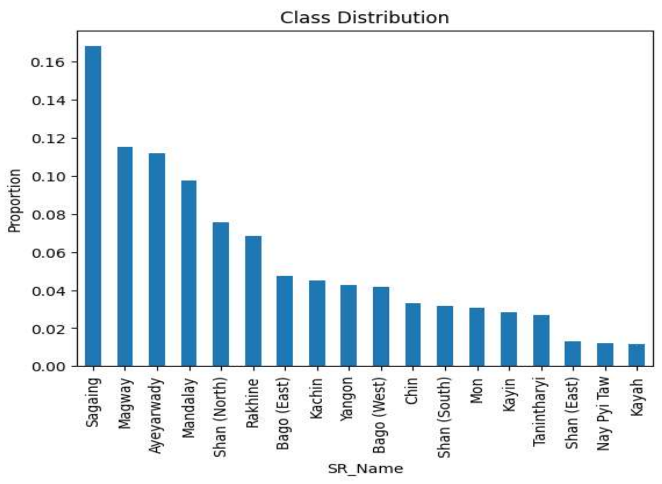


Fig 2: Regional Class Proportion in Dataset

Encoding and Tokenization Methods

A key consideration in this project is how to convert text-based Romanized village names into meaningful numerical representations for model input. Various encoding methods were explored:

1. **One-Hot Encoding (Baseline)**

Used as a baseline, one-hot encoding results in sparse representations and ignores character or semantic relationships. While simple to implement, it lacks the expressiveness needed for short, unique names. This model achieved relatively high training accuracy but failed to generalize (Test Accuracy: 28%).

2. **TF-IDF Consideration**

   TF-IDF was initially considered as a feature extraction method. However, it was ultimately not used because it is less effective for **very short texts** like village names, where meaningful term frequency patterns are limited or non-existent. Additionally, since the goal of this project is to build a deep learning model using TensorFlow, TF-IDF is not well-suited for integration with embedding layers and sequence-based architectures. Instead, token-based or character-level embeddings provide better compatibility and performance for this task.

3. **Character-Level Tokenization (Keras Tokenizer)**

This approach uses Keras's Tokenizer to convert each individual character into a unique integer. It is well-suited for handling **spelling variations** and **partial matches**, which are common in Romanized Burmese due to diverse ethnic pronunciations.

- **Example:**
  - Input: "Nay"
  - Character Tokens: ['n', 'a', 'y']
  - Tokenized Sequence: [1, 2, 3] (where each number maps to a character index)

This method allows the model to learn patterns at the **subword level**, making it more robust to inconsistent spellings.

4. **N-Gram Tokenization (TextVectorization Layer)**

The TextVectorization layer from TensorFlow was used to generate **character n-grams** (such as bigrams and trigrams). This approach captures **local character dependencies**, which can help the model understand meaningful combinations of characters.

- **Example (Bigrams):**
  - Input: "Nay"
  - Bigrams: ['na', 'ay']
  - Tokenized Sequence: [2, 3] (where each n-gram is mapped to a unique index)

N-gram tokenization enriches the input representation by preserving short character sequences, which are especially useful in modeling short texts like village names.

5. **FastText Embeddings**

FastText embeddings were trained on the dataset to capture meaningful representations of Romanized Burmese village names. Unlike traditional word embeddings, FastText represents each word as a combination of its **character n-grams**, allowing it to model **subword-level information**.

This is especially beneficial for this task, as Romanized Burmese names often have **non-standard or inconsistent spellings** due to variations in pronunciation across ethnic groups. By breaking words into smaller character sequences, FastText can generate useful embeddings even for **rare or unseen words**, improving the model's ability to **generalize and handle spelling variations** effectively.

Training Strategy and challenges

Given the **imbalanced nature of the dataset**, specific techniques were applied to reduce model bias toward overrepresented classes:

To address the **imbalanced dataset**, several strategies were applied:

- **Class Weighting:** The class_weight='balanced' option was used during training to give more importance to underrepresented classes.
- **Stratified Sampling:** train_test_split(..., stratify=y) ensured proportional class representation in both training and test sets.
- **Handling Ambiguous Labels:** Some village names appeared in multiple regions. These duplicates were temporarily removed as a **sanity check** to see if the model performs better without them. However, this led to information loss and would **not be recommended** in a real-world deployment.
- **Preventing Data Leakage:** Data was split before preprocessing to ensure that no test information leaked into the training process.
- **All the tested models used softmax activation for the output layer and relu for the dense layers.**

Table 1: Performance Comparison of Different Text Classification Models

Model ID	Architecture Description	Encoding Method	Training Accuracy (%)	Test Accuracy (%)	Remarks
Baseline	Input layer → 2 hidden layers → Output layer	One-hot encoding	84	28	Significant overfitting observed.

Model 1	Input → 3 hidden layers (with Dropout + BatchNorm after each) → Output	One-hot encoding	87	25	Regularization techniques did not mitigate overfitting.
Model 2	Embedding → BiLSTM → GlobalMaxPooling → Dropout → Dense → Dropout → Output	Character-level tokenization	60	25	LSTM underperforms with char-level features.
Model 3	Embedding → BiLSTM → Dropout → BiLSTM → GlobalMaxPooling → Dropout → Dense → Dropout → Dense → Dropout → Output	Character-level tokenization	45	22	Deep recurrent structure worsens performance, likely due to vanishing gradients or data sparsity.
Model 4	Embedding → Conv1D → MaxPooling → BiLSTM → GlobalMaxPooling → Dense → Dropout → Dense → Dropout → Output	character-level (0-gram) tokenization	40	27	Mixed architecture did not improve generalization.
Model 5	Conv1D → LSTM	character-level (bigram) tokenization	77	26	Shows better training accuracy but still overfits.
Model 6	Conv1D → LSTM	character-level (trigram) tokenization	69	24	Increasing n-gram complexity leads to performance degradation.
Model 7	Embedding→ Conv1D →GlobalMaxPool→ →Dense(1024) → Dropout → Dense(512) → Dropout → Dense(64) → Dropout → Output LSTM (Epochs = 60)	FastText embeddings	95	38	Strong training and improved test accuracy; FastText shows promise.
Model 8	Embedding→ Conv1D →GlobalMaxPool→ →Dense(1024) → Dropout → Dense(512) → Dropout → Dense(64) → Dropout → Output LSTM (Epochs = 100)	FastText embeddings	92	40	Further training improves generalization marginally.
Model 9	**Embedding→ Conv1D →GlobalMaxPool→ LSTM →Dense(1024) → Dropout → Dense(512) → Dropout → Dense(64) → Dropout → Output (Epochs = 130)**	**FastText embeddings**	**95**	**40**	**Best test accuracy so far.**

Model 10	Embedding→ Conv1D →GlobalMaxPool→LSTM →→Dense(1024) → Dropout → Dense(512) → Dropout → Dense(64) → Dropout → Output (Epochs = 80)	Phonetic Normalization + FastText embeddings	92	35	Phonetic normalization didn't help a lot.
Model 11	Embedding→ Conv1D →GlobalMaxPool→ →Dense(1024) → Dropout → Dense(512) → Dropout → Dense(64) → Dropout → Output (Epochs = 130)	Phonetic Normalization + FastText embeddings	95	37	Slight drop in test accuracy.

Observations:

- Deep LSTM-based models struggled, possibly due to limited training data or the inability to extract meaningful long-term dependencies from very short inputs.
- Character n-gram tokenization with convolutional models performed moderately.
- FastText-based Wide ConvNets outperformed all other models in generalization ability, likely due to better handling of subword patterns in Romanized place names.
- **Training Time:** All tested models had reasonable training times, ranging between **100 to 250 seconds**, making the experiments manageable and efficient.

Despite these efforts, **test accuracy remained low** and **overfitting persisted**, likely due to:

- **Short and sparse inputs:** Village names are often too short and unique, limiting the model's ability to learn patterns.
- **Weak signal:** There may be little direct correlation between name and region in many cases.
- **Imbalanced and limited data:** Some classes lack sufficient diversity or representation.

Continued experimentation with more **robust embeddings**, **data augmentation**, and **hybrid models** may be necessary to further improve performance.

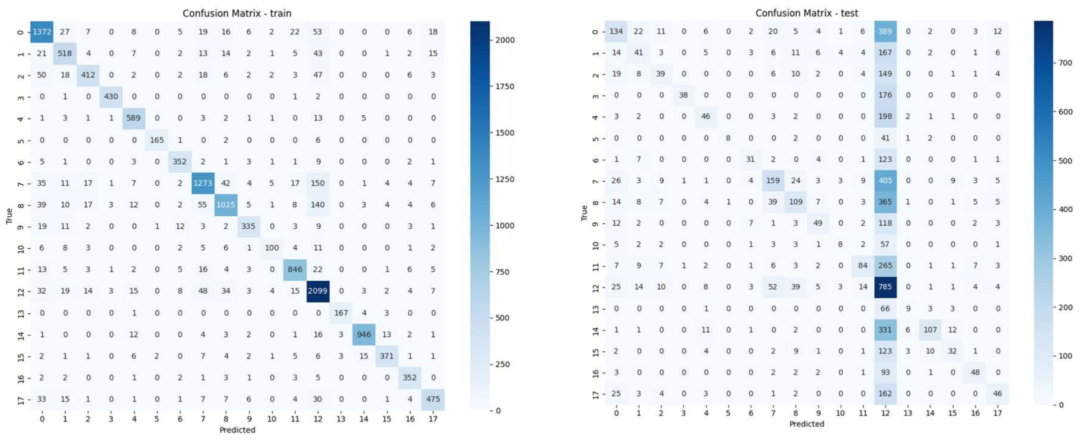


Fig 3: baseline confusion matrix for train and test

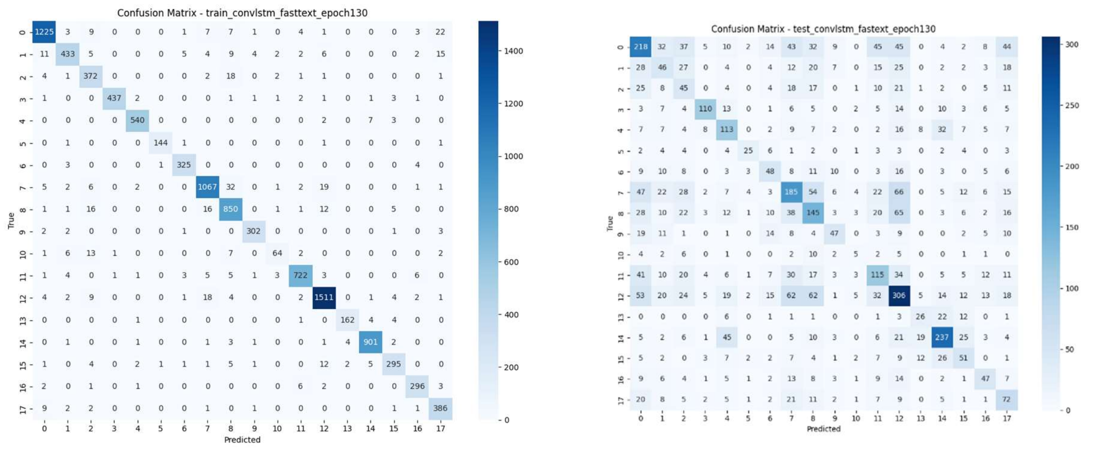


Fig 4: best model confusion matrix for train and test

Confusion Matrix Analysis

Figures 3 and 4 show the confusion matrices for the **baseline model** and the **best-performing model** (Conv1D + LSTM with FastText embeddings, trained for 130 epochs).

- **Baseline Model**:
  The training matrix shows strong class-wise learning, but the test matrix reveals **severe overfitting**, with predictions highly biased toward a few dominant classes (e.g., classes 11 and 13). Misclassifications are widespread across less-represented classes, resulting in poor generalization.
- **Best Model**:

  In contrast, the best model demonstrates a more balanced prediction pattern on both train and test data. While some confusion remains for certain minority classes, test performance significantly improves—especially in terms of class coverage and prediction distribution. The model better differentiates between classes and shows reduced bias toward dominant labels.

  Although the test accuracy remains relatively low, the confusion matrix reveals that the model is making more meaningful predictions across a wider range of classes, rather than collapsing to a few frequent ones. This suggests improved learning and generalization, even if overall accuracy still needs improvement.

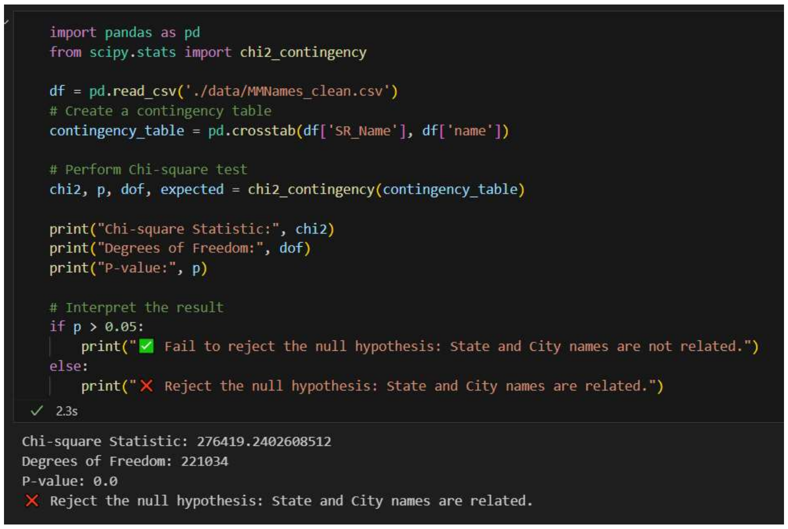
Key Improvements Observed:

- **Better generalization**: Less overfitting compared to the baseline.
- **Wider diagonal dominance**: Indicates stronger and more consistent correct predictions across classes.
- **Improved minority class recognition**, reflecting enhanced robustness.

Statistical Justification

To verify whether village names are meaningfully linked to their regions, a **chi-square test of independence** was conducted. The test returned a **p-value < 0.05**, allowing us to **reject the null hypothesis**. This confirms a **statistically significant relationship** between village names and their corresponding regions.

This insight supports the core idea of the project: **it is feasible to build a predictive model** for region classification based on village names—despite the challenges posed by short and diverse inputs.



```
import pandas as pd
from scipy.stats import chi2_contingency

df = pd.read_csv('./data/MMNames_clean.csv')
# Create a contingency table
contingency_table = pd.crosstab(df['SR_Name'], df['name'])

# Perform Chi-square test
chi2, p, dof, expected = chi2_contingency(contingency_table)

print("Chi-square Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-value:", p)

# Interpret the result
if p > 0.05:
    print("✅ Fail to reject the null hypothesis: State and City names are not related.")
else:
    print("❌ Reject the null hypothesis: State and City names are related.")
```

✓ 2.3s

```
Chi-square Statistic: 276419.2402608512
Degrees of Freedom: 221034
P-value: 0.0
❌ Reject the null hypothesis: State and City names are related.
```

Fig 5: chi2_contingency test

Conclusion

This project explored the challenge of classifying Myanmar village names into their respective states and regions using a range of deep learning architectures and encoding techniques. While the problem seems simple at first glance, it is complicated by several factors: **very short and ambiguous inputs**, **class imbalance**, and **village names appearing in multiple regions**.

After testing multiple models, the best performance was achieved using **FastText embeddings combined with deep convolutional layers**, reaching a **maximum test accuracy of 40%**. This confirms that **subword-level information**, as captured by FastText, is helpful in generalizing over non-standard spellings.

Despite this improvement, **most models showed significant overfitting**, and **generalization remained limited**, especially in models using one-hot or character-level encodings alone. These results suggest that while a **statistically significant relationship** exists between village names and their regions (as supported by the chi-square test), current deep learning approaches struggle to fully capture it due to:

- Lack of contextual information in the input,
- Ambiguity in name-region mappings,
- Limited training data for underrepresented regions.

Further improvements could involve **richer contextual features**, **data augmentation**, or **hybrid models** that combine deep learning with rule-based or geographic heuristics.

Recommendations for Future Work

To address the limitations observed in this project and enhance model performance, the following strategies are recommended:

1. Data Quality and Expansion

- Collect more labeled data across all regions to reduce class imbalance and improve generalization.
- Standardize Romanized spellings to minimize variation caused by ethnic or individual transliteration styles.
- Instead of removing duplicate village names tied to multiple regions, consider framing the problem as a **multi-label classification** task, where applicable.

2. Feature Engineering and Enrichment

- Integrate additional features such as **common suffixes**, **geographical proximity**, or **region-specific prefixes** to provide contextual cues.
- Experiment with **phonetic encodings** (e.g., Soundex, Metaphone) to group names with similar pronunciations.

3. Advanced Modeling Approaches

- Explore **transformer-based models** (like BERT or RoBERTa) pre-trained on Burmese or short-form Romanized text for better handling of context and variation.
- Build **hybrid models** combining CNNs (for local patterns) and LSTMs/transformers (for sequence understanding).

4. Handling Class Imbalance

- Apply **text augmentation techniques** to generate synthetic samples for underrepresented regions.