CA1 - PartA

Convolutional Neural Network

Problem Statement:

Predicting Vegetables

May 20 2025

Kaung Myat San, P2408655

Agenda

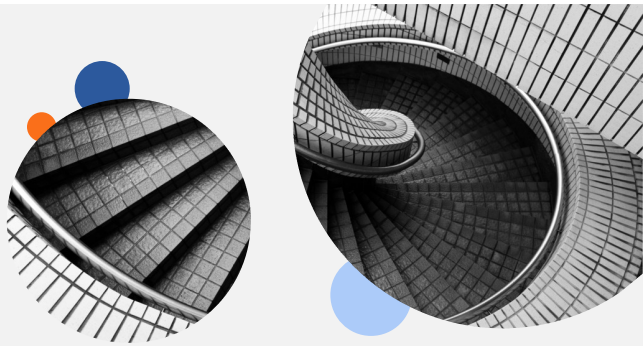
- Background Research & Data Analysis
- Data Augmentation
- Baseline Model
- Hyper-Tuning
- Improving Hyper-Tuned


Background
Research & Data
Analysis

Dataset Overview

- Image Size: 224 x 224 pixels (3 color channels- rgb)
- Number of class: 11 classes (13 vegetables)
- Classes:
  - Bean
  - Bitter Gourd
  - Bottle Gourd and Cucumber
  - Brinjal
  - Broccoli and Cauliflower
  - Cabbage
  - Capsicum
  - Carrot and Radish
  - Potato
  - Pumpkin
  - Tomato

Data Analysis



Irregularities

- Test Folder:
  - Pumpkin and Tomato Folders are swapped
- Train Folder:
  - Bean Folder: 0001-0020, 0049, 0050.jpg are carrots

How to fix

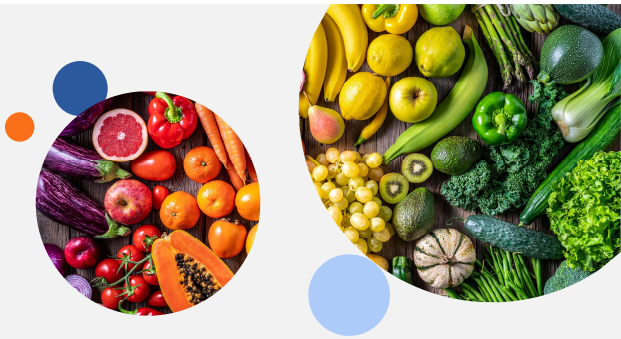
- Assigned it as the correct label when processing and loading image
- Ensure correct label for each vegetable folder

Metrics to Evaluate

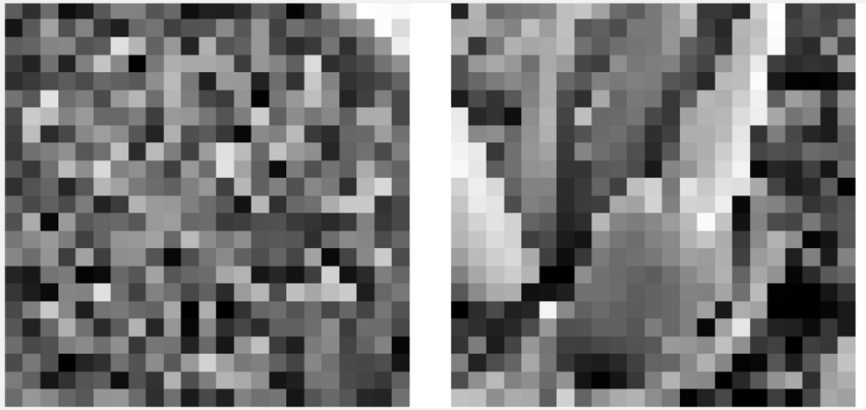
- **Loss –** metric to monitor during training
- **Recall** – Reduce false negatives (catch all actual vegetables)
- **Accuracy** – Overall Correct Predictions
- **Precision** – Reduce false positives (avoid wrong vegetable predictions)

- **Focus:** Loss, Recall & Accuracy
- **Reason:** shows a complete view of the model's performance

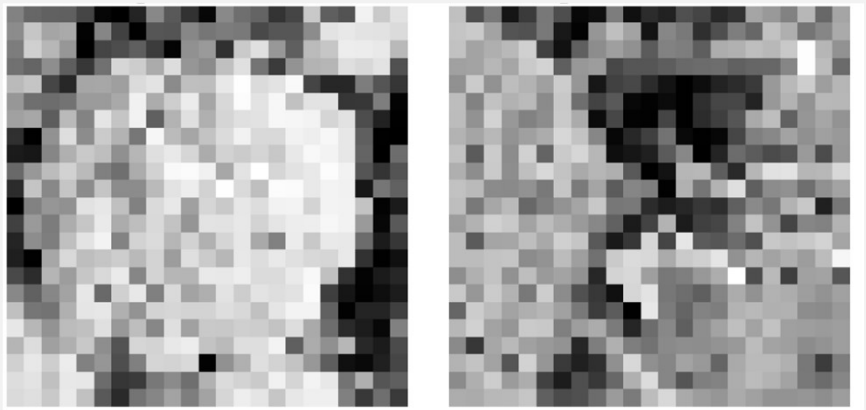
Why Group Some Vegetables?



Bottle Gourd & Cucumber



Broccoli & Cauliflower



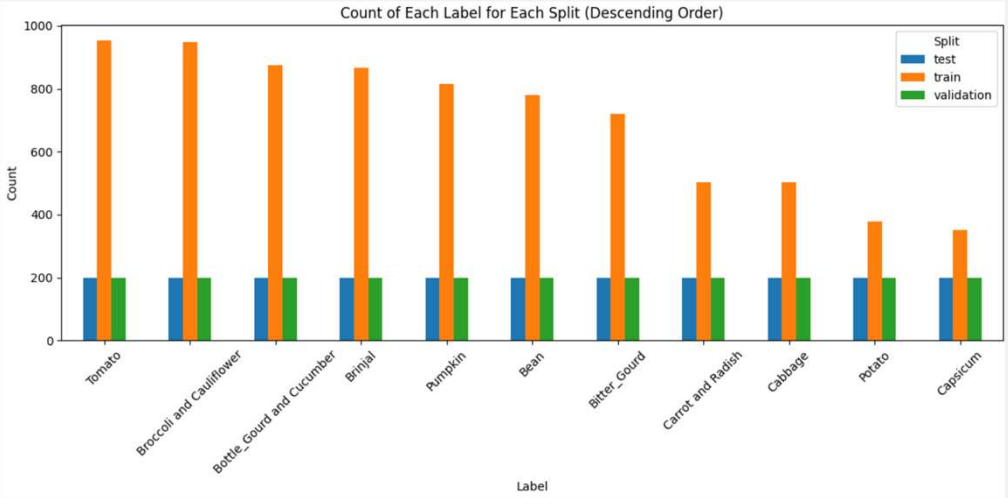
- Can you differentiate between these 2 vegetables?
- These 2 vegetables looked very similar with gray-scaled and challenging for our model
- Therefore, these vegetables will be grouped together

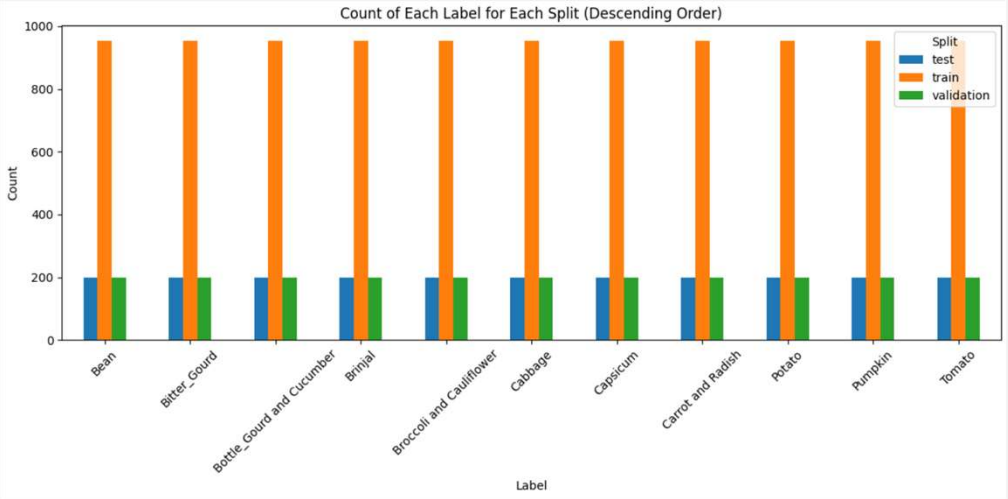

Data
Processing

Data Processing

- Convert to gray-scale and corresponding size
- Convert to array
- Normalized the array
- Convert into one-hot encoding
- Removed duplicates – 17

- Perform Data Transformation to create a more balanced dataset
- Generate Images so all classes have same amount as the max.





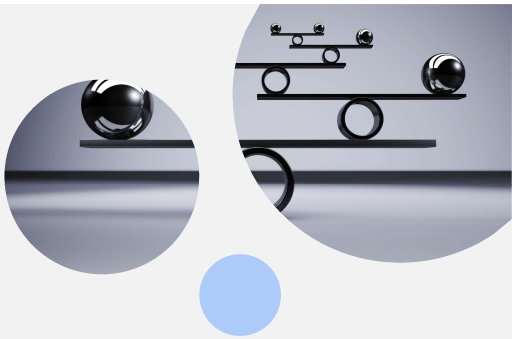

Baseline
Model

Base Model Layers

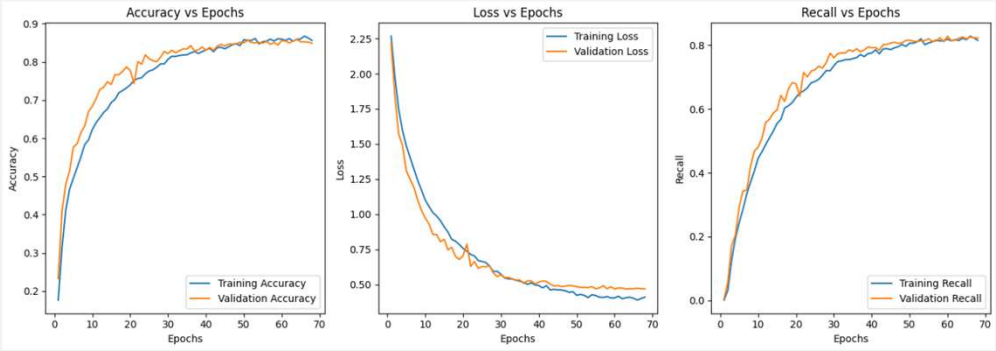
- Compile
  - optimizer = Adam(learning_rate = 0.001)
  - loss = “categorical_crossentropy”
- Fitting
  - Early Stopping - monitor= val_loss, patience = 5, restore_best_weights = true
  - reduce_lr – monitor = val_loss, factor = 0.5, patience = 3, min_lr = 1e-6
  - epochs = 100
  - batch_size = 32

Layers	Parameters
Conv2D	Filters = 32
MaxPooling2D	pool_size = (2,2)
Dropout	0.25
Conv2D	Filters = 64
MaxPooling2D	pool_size = (2,2)
Dropout	0.25
Flatten	0.25
Dense	128, ‘relu’
Dropout	0.5
Dense	11, ‘softmax’

Balanced Vs Imbalanced (101x101)

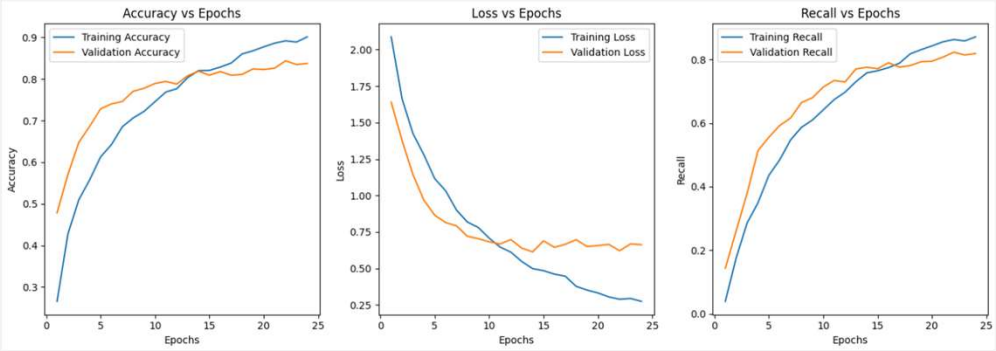


Balanced



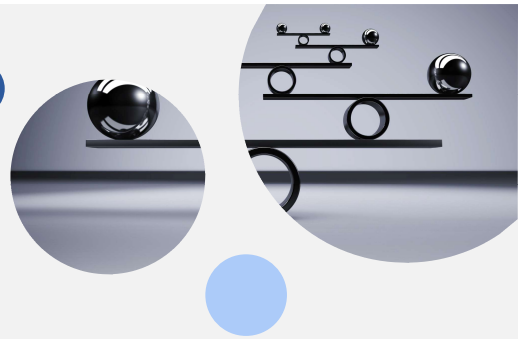
- Precision: 86%
- Recall: 85%
- Accuracy: 85.23%

Imbalanced

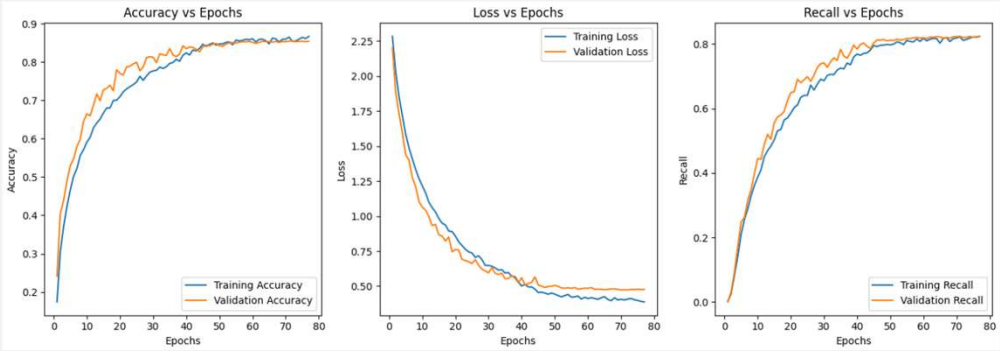


- Precision: 82%
- Recall: 82%
- Accuracy: 82.23%

Balanced Vs Imbalanced (23x23)

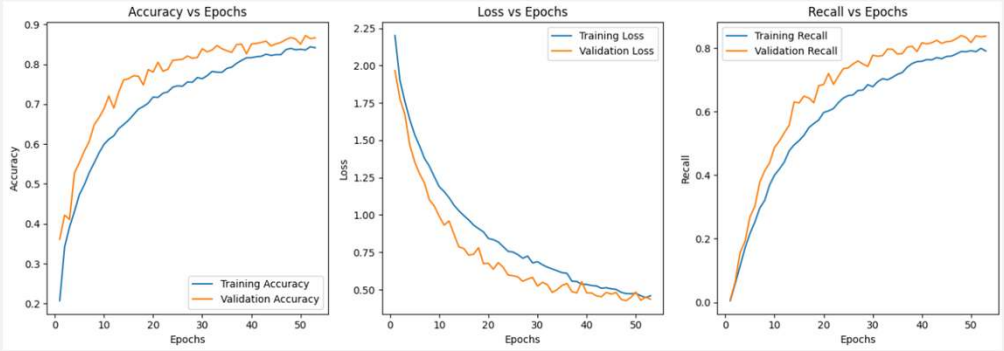


Balanced

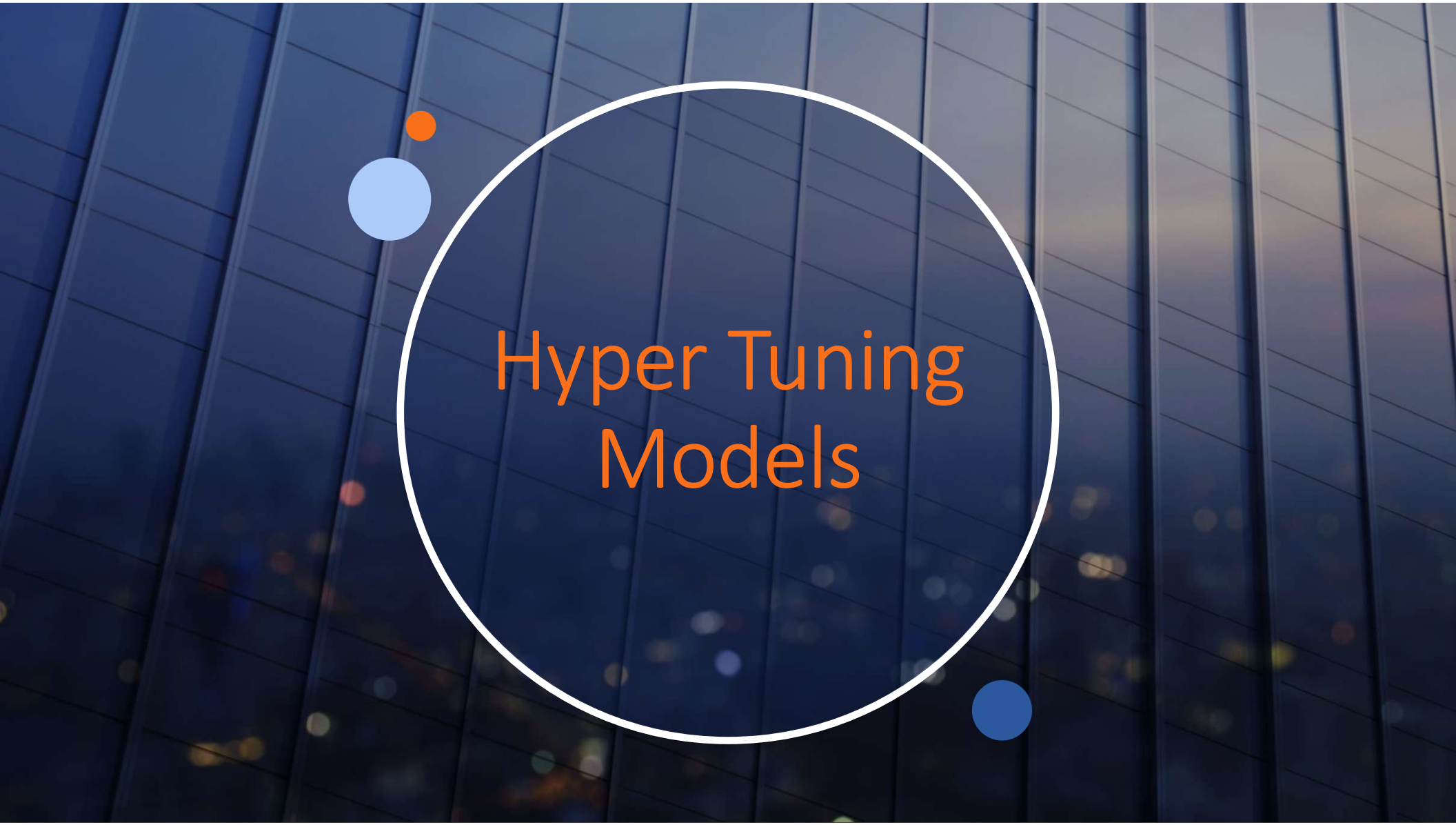


- Precision: 87%
- Recall: 86%
- Accuracy: 86.27%

Imbalanced



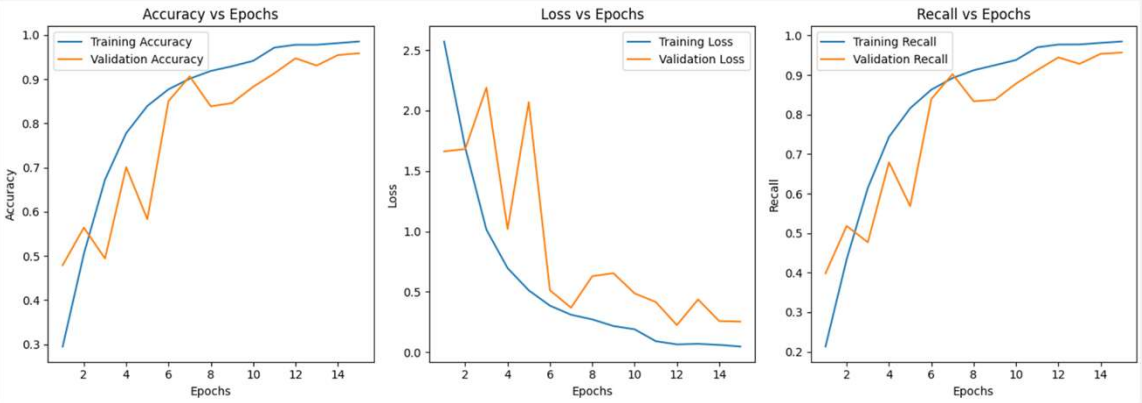
- Precision: 87%
- Recall: 86%
- Accuracy: 86.41%

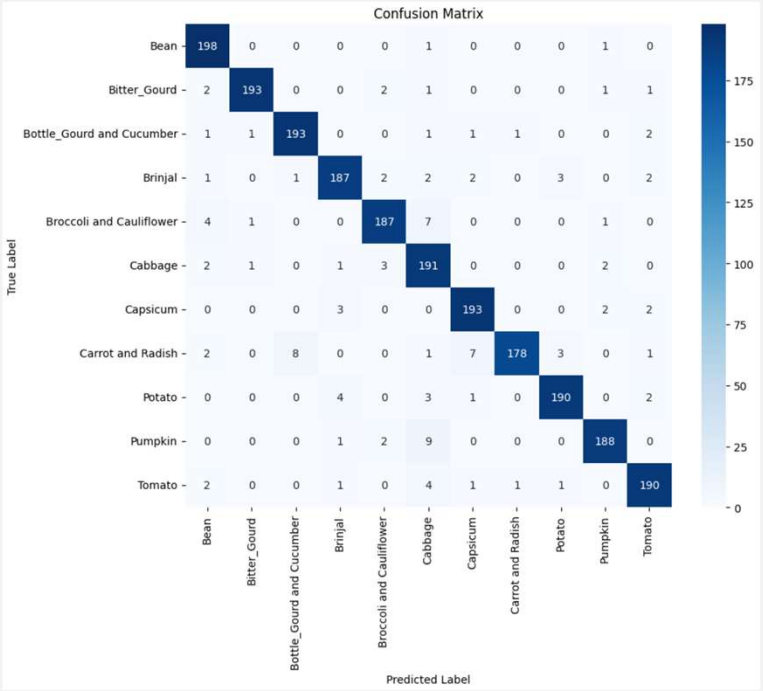

Hyper Tuning
Models

Hyper Tuning Parameters

Parameters	Details	Parameters	Details
Conv_filters	The order of dimension of filters in Conv2D	final_activation	Activation function for the output layer
kernel_size	The order of list of convolutional windows	optimizer	Optimizer to use for compiling the model
pool_size	Pool size for MaxPooling2D layers	batch_norm	Whether to include BatchNormalization layers
dense_units	Number of units in the dense layer	learning_rate	Learning rate for the optimizer
dropout_rates	List of dropout rates for each Dropout layer	no_of_conv_layers	Number of Conv2D layers to stack for each filter size
activation	Activation function for Conv2D and Dense layers	~~l2_regularization~~	L2 regularization factor for Conv2D layers

101x101 Best Hyper Tuned Model



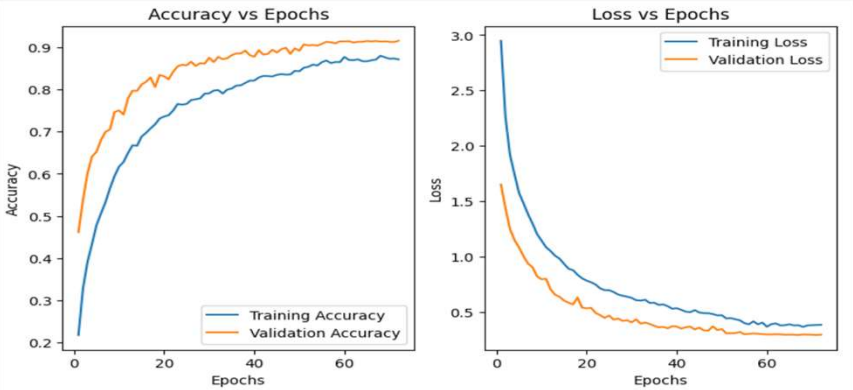


- Test Recall: 0.95
- Test Precision: 0.95
- Test Accuracy: 0.95

Area for Improvement

- **Over-Fitting:** Reduce dropout slightly
- **Misclassification:** Augment more data for Poorly Classified Class

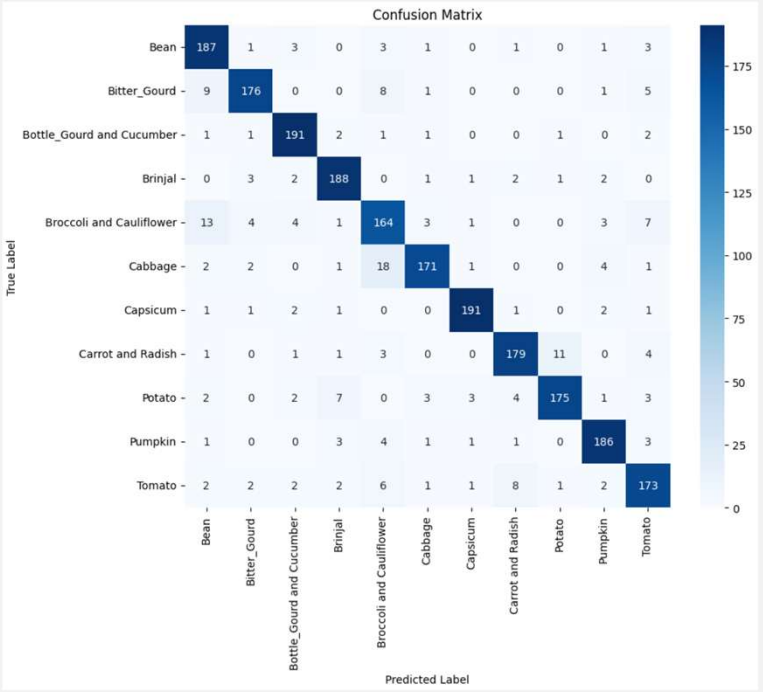
23x23 Best Hyper Tuned Model



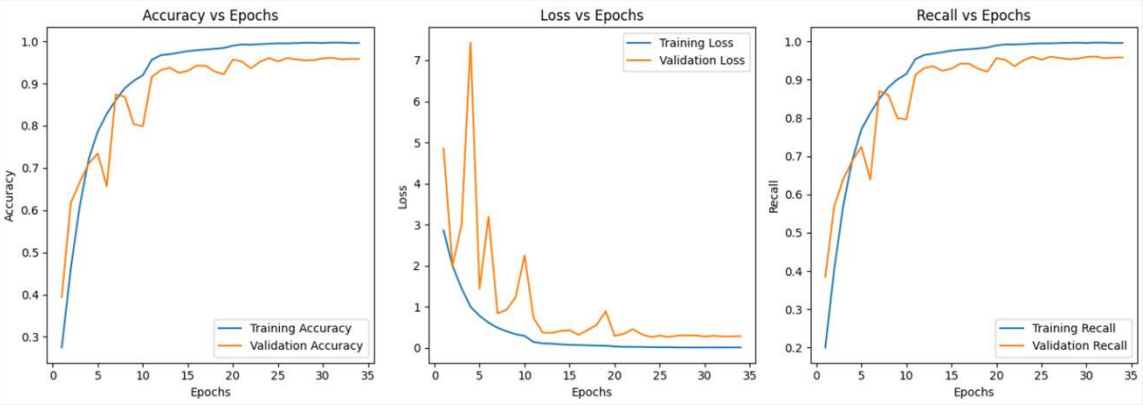
- Test Recall: 0.90
- Test Precision: 0.90
- Test Accuracy: 0.90

Area for Improvement

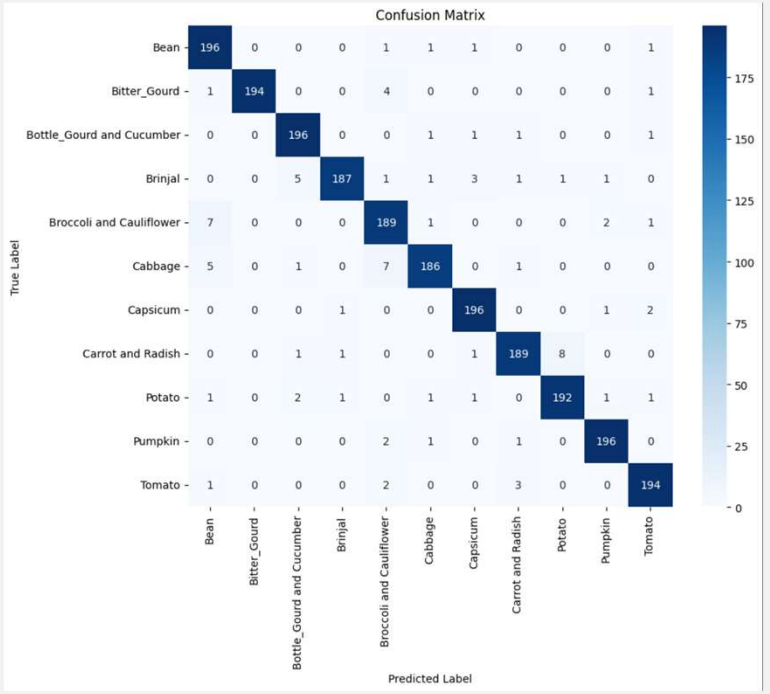
- **Misclassification:** Augment more data for Poorly Classified Class
  1. Broccoli & Cauliflower
  2. Cabbage
  3. Tomato
  4. Potato
  5. Bitter Gourd & Cucumber
  6. Bean



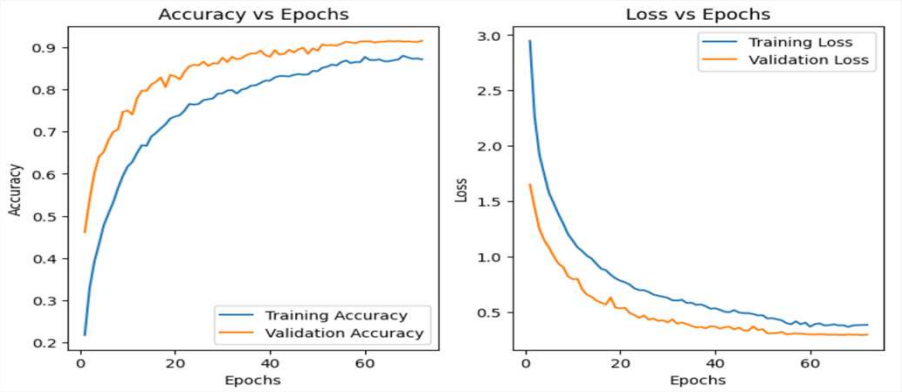
101x101 Final Best Model

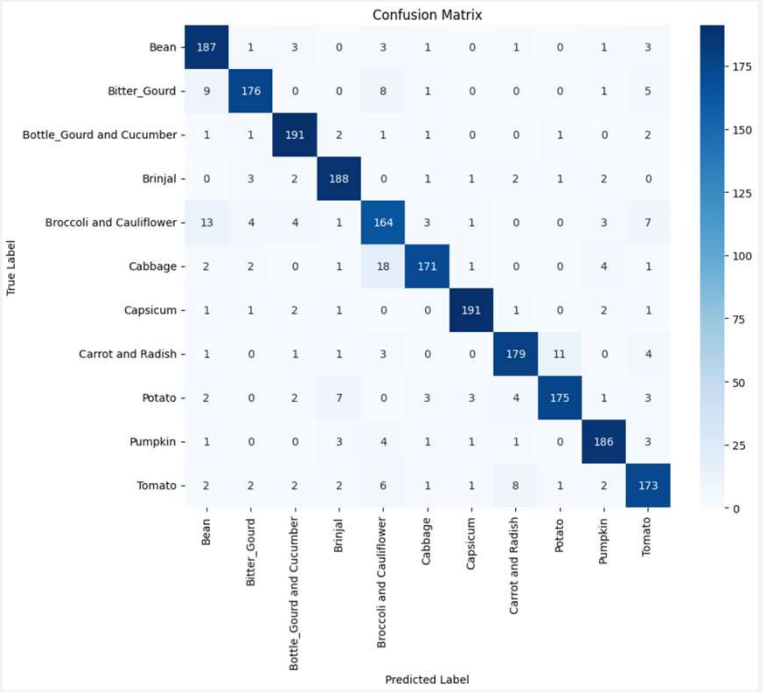


- Test Recall: 0.96
- Test Precision: 0.96
- Test Accuracy: 0.96



23x23 Final Best Model





- Didn't Improve after trying Data Augmentation and Data Removal based on Confusion Matrix
- Reason: Augmenting a 23 by 23 images might cause them to lose distinctive features, making different vegetables similar
- Test Recall: 0.90
- Test Precision: 0.90
- Test Accuracy: 0.90

CA1 - PartB

Recurrent Neural Network

Problem Statement:

Predicting Movie Review's Sentiment Score

May 20 2025

Kaung Myat San, P2408655

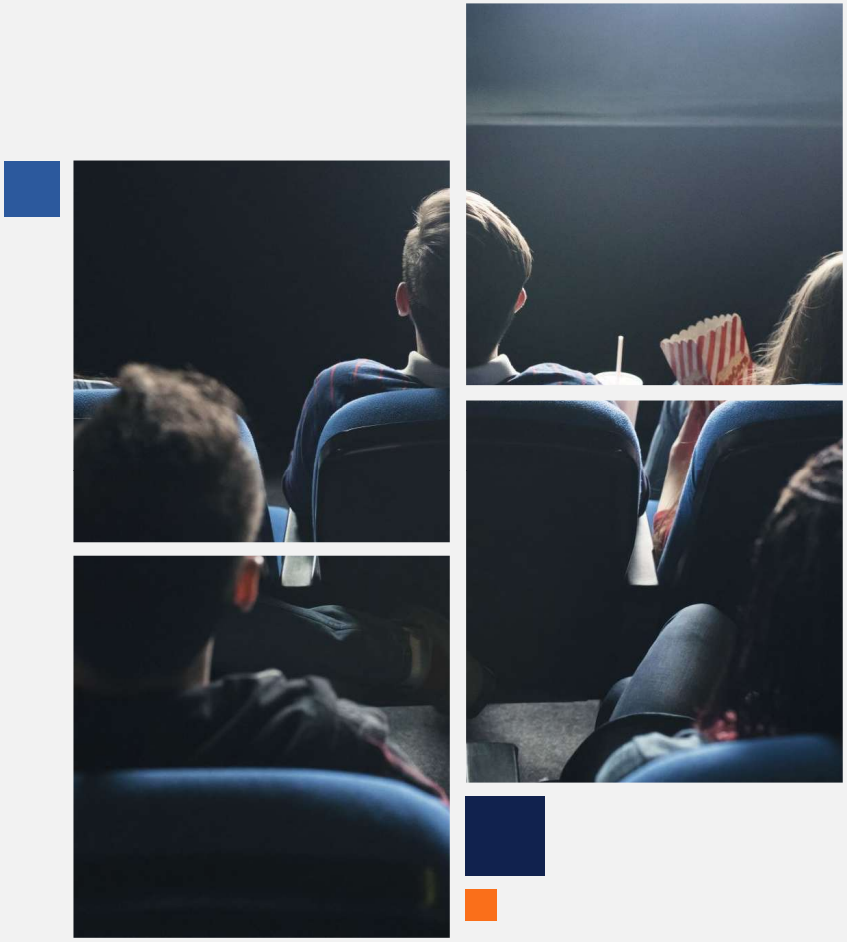
Agenda

- Background Research & Data Analysis
- Data Augmentation
- Text Preprocessing
- Custom Embedding
- Baseline Model
- Hyper-Tuning
- Conclusion

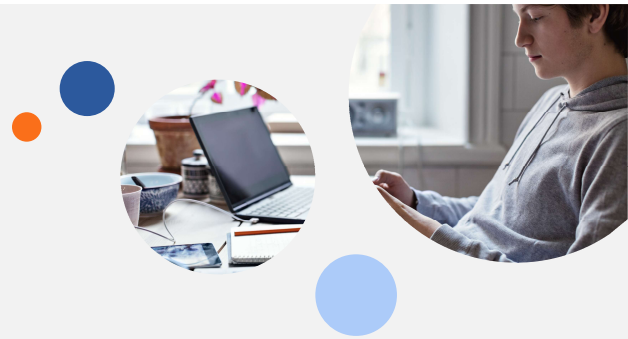

Background
Research & Data
Analysis

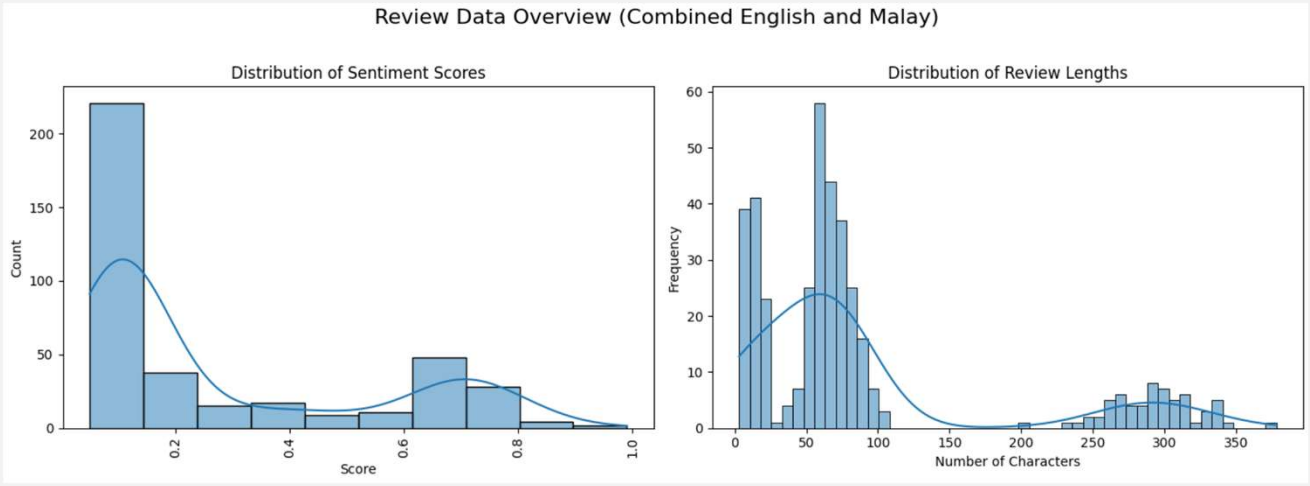
Dataset Overview

- Movie Review Dataset contain 4 columns:
  - **Review**: Review of the movie
  - **Score**: Associated Sentiment Score for each Review
  - **Are there ways for you to generate more data? Splitting up sentences, would that help?:** null
  - **Language:** the language of each review
- **Sentiment Score Range:** 0-1 (positive to negative)
- **No of Language:** 4 (mostly Malay & English)
- **Review**: Convert to English for Normalization and easy interpretation



Dataset Insights





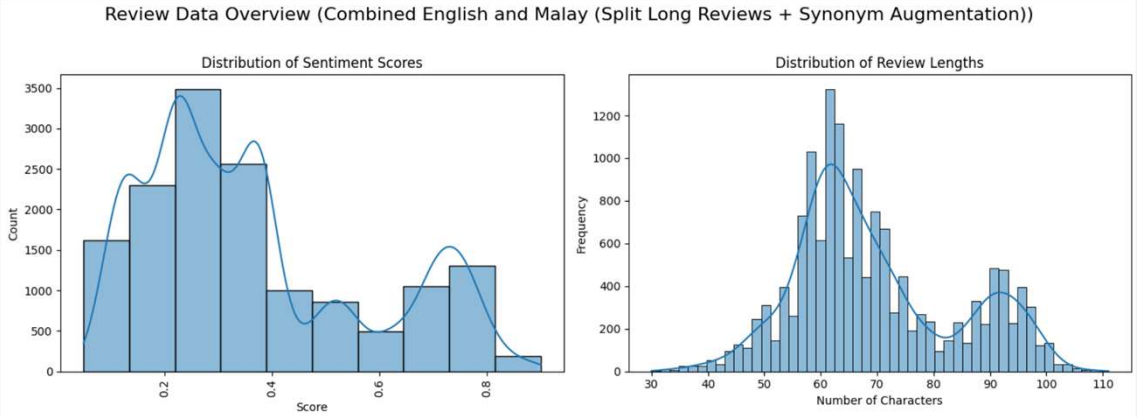
- Both: Bimodal Distribution, Right-Skewed
- Scores: around 4 times positive as negative
- Review Length: 3 groups – very short, short and long reviews
- Long Reviews: multiple sentences (can split)
- Very Short Reviews: 1, 2 or 4 words (can be used to swap words in Short Reviews)
- Short Reviews: Good for predicting


Data
Augmentation

Reviews Word Swapping Using Reviews (< 30 characters)

- Reviews (< 30 char) follow a pattern
  - “Amazing”, “Amazing Action”
  - “Breathtaking”, “Breathtaking Visuals”
- Steps to Swap Reviews
  1. Extract Word List with Sentiment Score
  2. Create a Phrase Dictionary where each base word (eg: Action) has a list of modifiers (eg: Amazing)
  3. Find Base Word in Reviews
  4. Swap Modifier if there is a Modifier else Add Modifier
  5. Add this swapped Review as new Rows
  6. Repeat for all Reviews
- New Row Score: $\frac{Original\ Score + Scor\ \ of\ Swapped\ Out + Score\ of\ Swapped\ In}{3}$
- Remove Duplicates

Replace Synonym of Adjective in Reviews



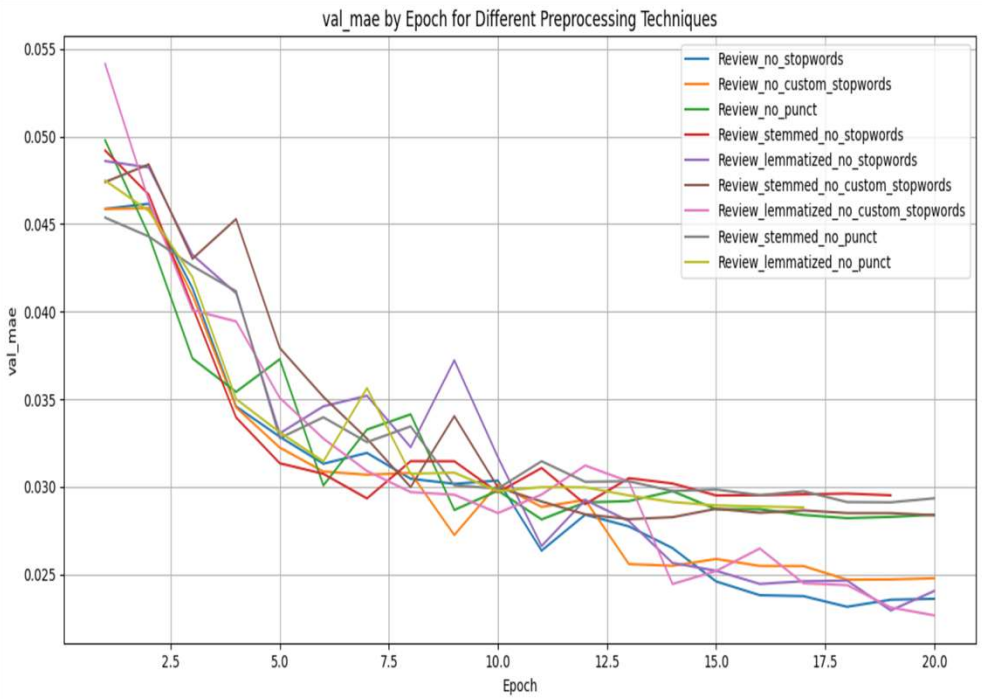
- Create bins and check count in each bins
- To balance dataset, only augment a certain amount in each bin
- Create a new Review with Replaced Synonym
- New Row Score: same score as original
- Remove Duplicates

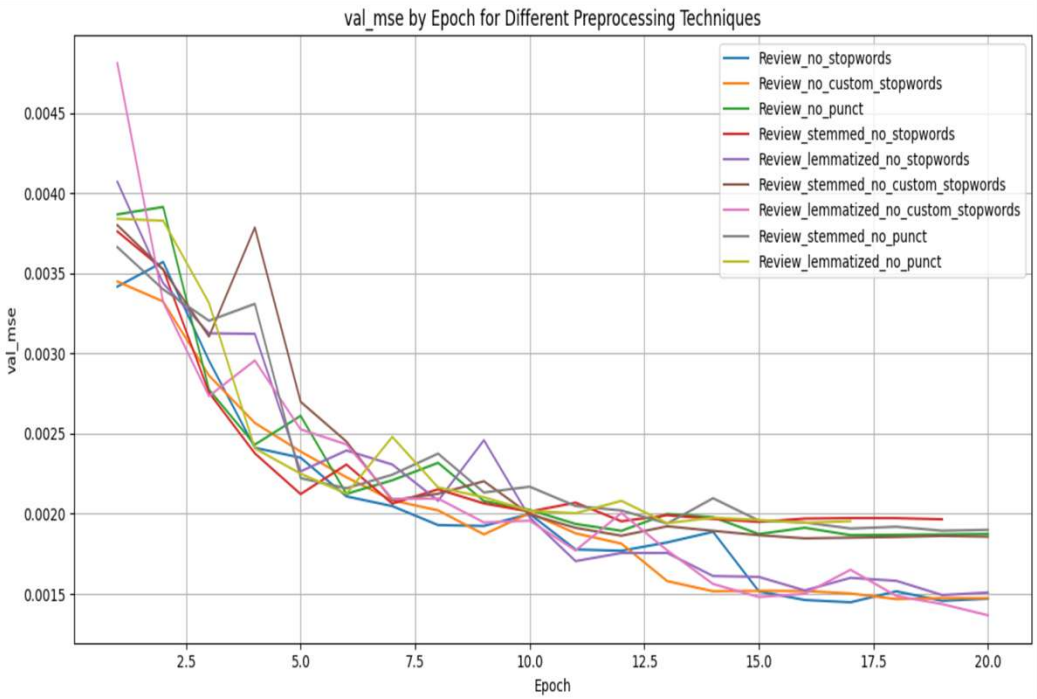
- Steps to replace Adjective with Synonym
  1. Check if words in Review are adjective
  2. If adjective, find synonym
  3. Return a random synonym
  4. Swap these 2 words
  5. Add this swapped Review as new Rows
  6. Repeat for all Reviews unless amount in bin is reached


Text
Preprocessing

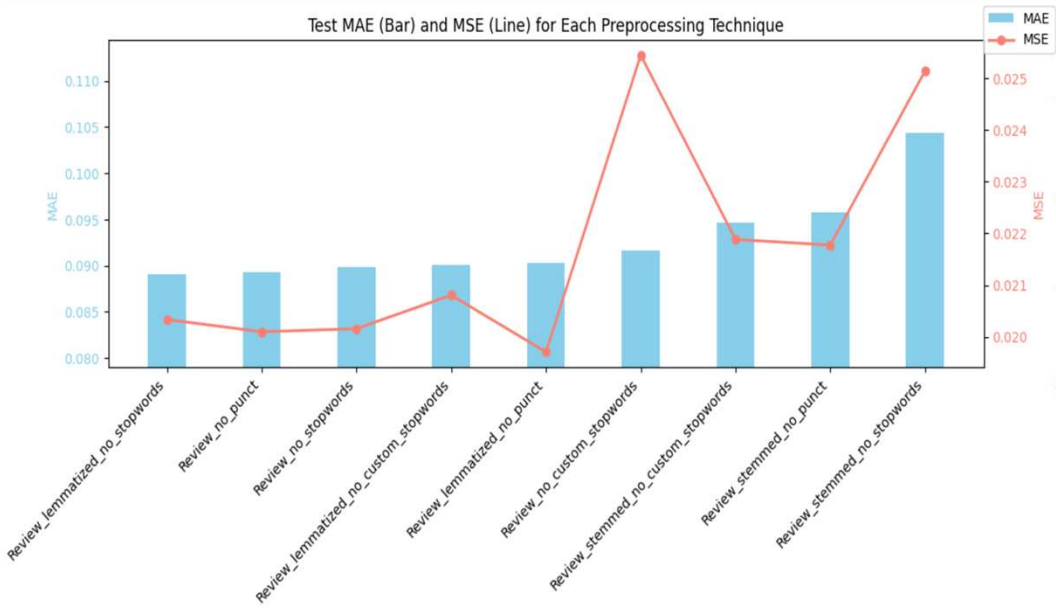
Different Text Preprocessing

- All Text Preprocessing have already **removed contractions**.
- Preprocessing that have **removed stop words** have also **removed punctuation**.





Best Text Transformation



Transformation	Val Performance	Test Performance
Review_lemmatized_no_stopwords	Very Good	Best
Review_no_punct	Ok	Very Good
Review_lemmatized_no_custom_stopwords	Best	Good

- Best Text Preprocessing:
  Review_lemmatized_no_stopwords
- Reason: Best overall preprocessing
- Best Test Performance , Very Good Val Performance


Custom
Embedding

Custom Embedding Weights

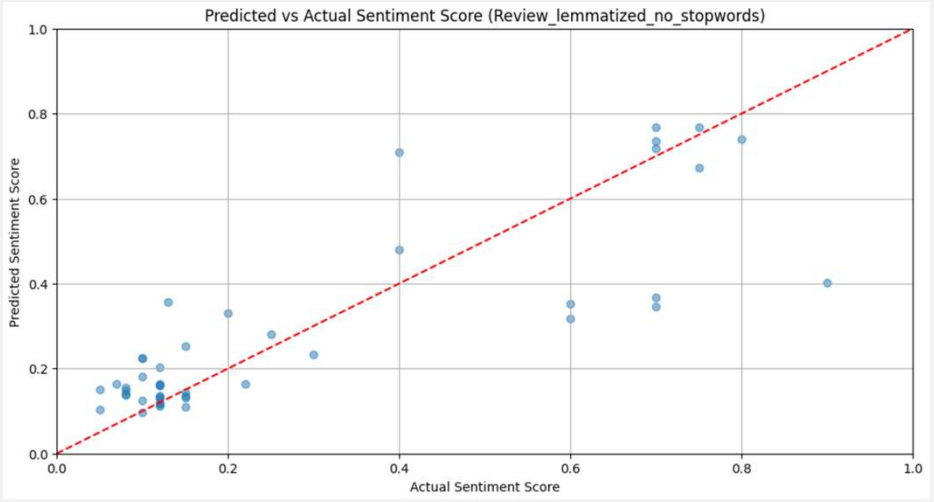
- Words in Reviews
  - 700 words appeared only in positive reviews
  - 700 words also appeared only in negative reviews
  - Remaining 400 words appeared in both context
- Steps to Create Weights
  1. create a DataFrame where each word has a count of where it appeared positive or negative
  2. tokenize each words to be able to assign weights
  3. Create a embedding matrix (no of unique words x embedding dim)
  4. Calculate embedding weights
  5. Assigned weights to first col and for rest add a little noise if **embedding dim** > 1
  6. Repeat for all words
- Embedding Weights Formula (-1, 1): $\frac{Count\ of\ Neg - Coun\ \ of\ Pos}{Count\ of\ Neg + Coun\ \ of\ Pos + \ .00001}$


Baseline
Model

Baseline Model

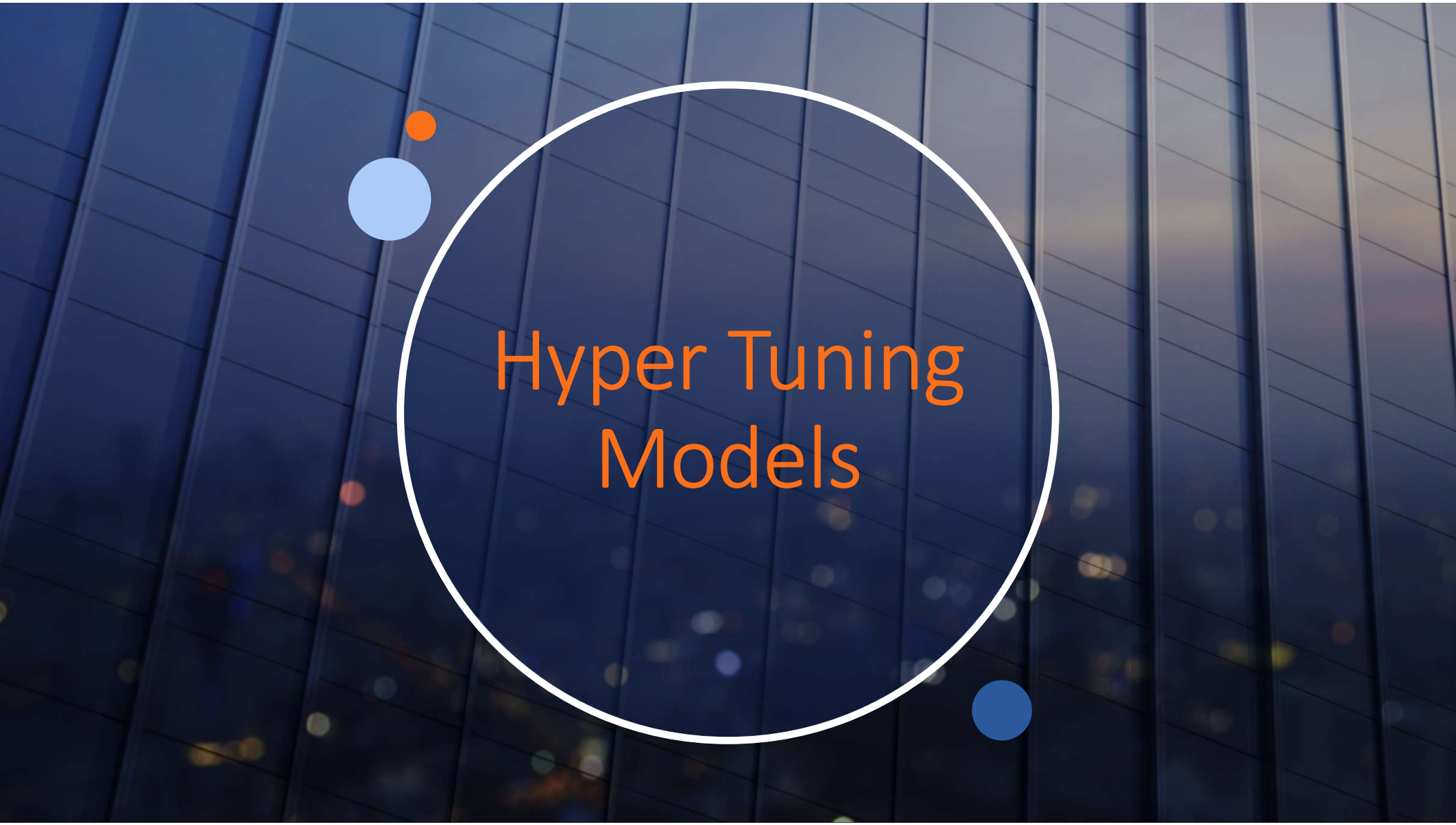
- Compile
  - optimizer = Adam(learning_rate = 0.001)
  - loss = “mse”
- Fitting
  - Early Stopping - monitor= val_loss, patience = 4, mode = “min”
  - reduce_lr – monitor = val_loss, factor = 0.2, patience = 3, mode=“min, min_lr = 1e-6
  - validation_split = 0.2
  - epochs = 100
  - batch_size = 32

Layers	Parameters
Embedding	weights = custom weight
LSTM	units=64, return_sequence = False
Dense	units=1 , activation=“sigmoid”



Test Metrics

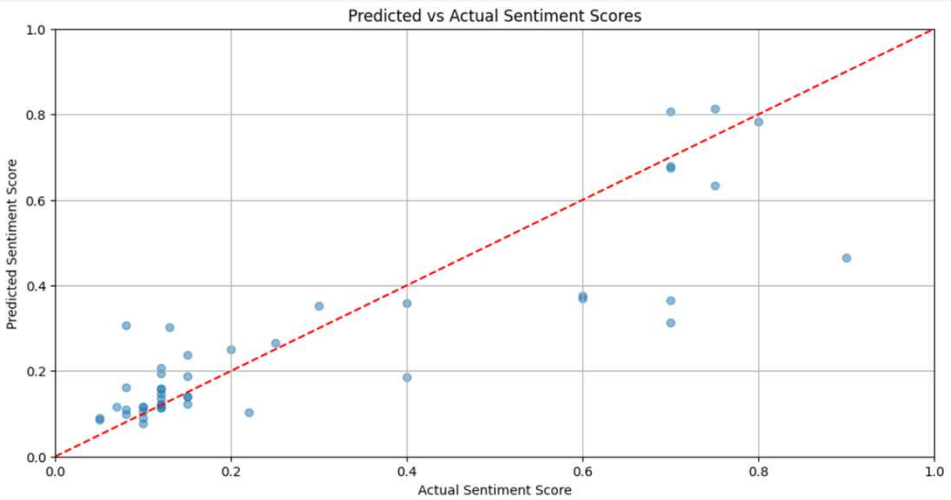
- MAE: 0.0934
- MAPE: 47.6619
- MSE: 0.0206
- MSLE: 0.0097

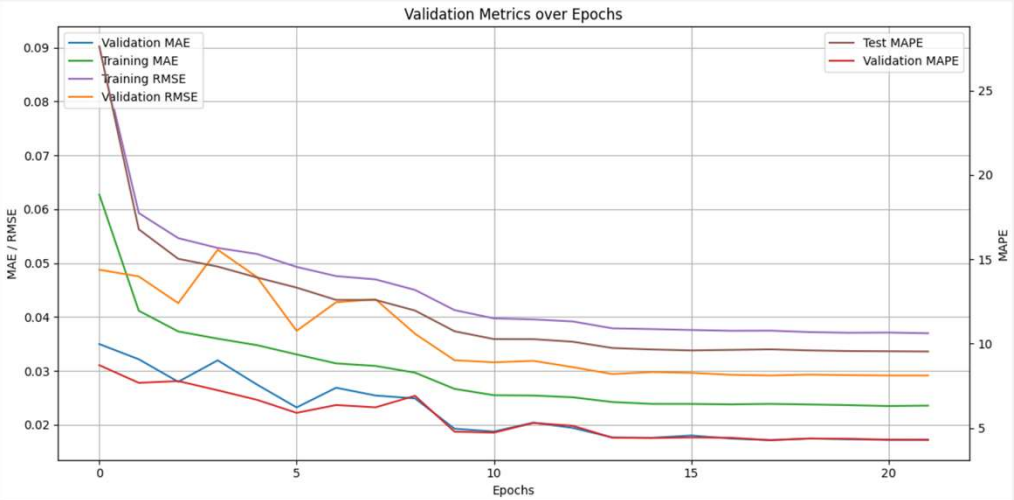

Hyper Tuning
Models

Hyper Tuning Parameters

Parameters	Details	Parameters	Details
column_name	name of the column (for vocabulary and embeddings)	dropout_layers	dropout rates for each RNN layer
rnn_type	'LSTM', 'GRU', or 'SimpleRNN'	batch_norm	whether to apply batch normalization
embedding_dim	output dimension of the embedding layer	optimizer	keras optimizer instance
rnn_units	number of RNN units	bidirectional	whether to use bidirectional RNN
No_of_RNN_layers	number of RNN layers	dense_units_list	units for each Dense layers

Best Hyper Tuned Model



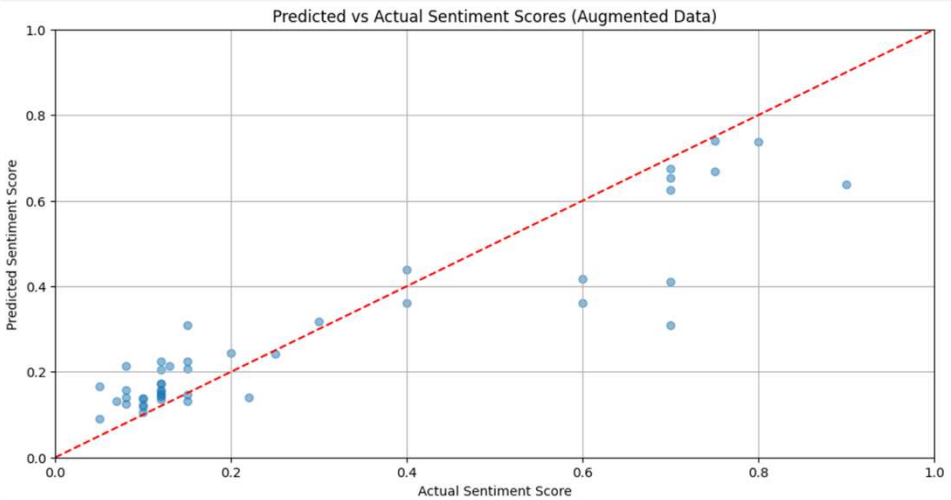


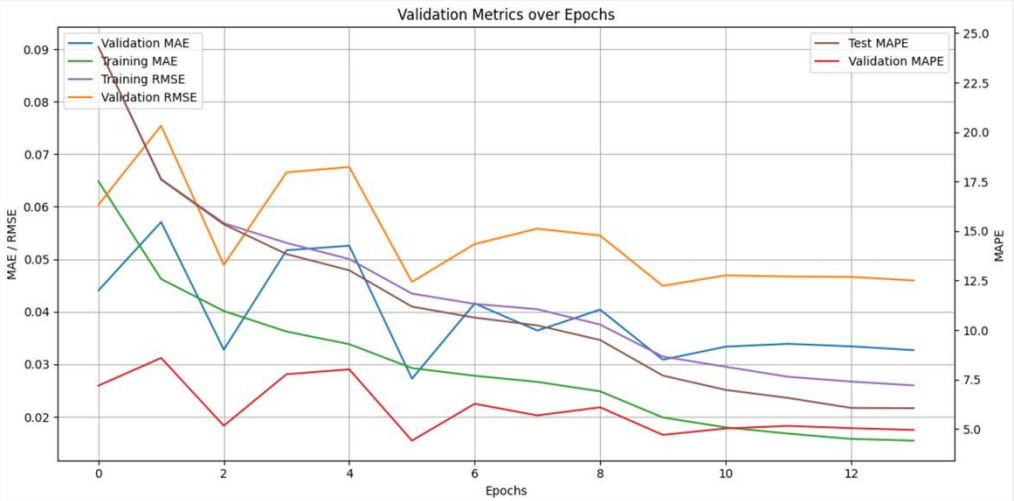
- Test MAE: 0.0808
- Test MSE: 0.0172
- Test RMSE: 0.1311
- Test MSLE: 0.0082

Model Insights

- **Performance**: Great (Low Error Metrics)
- **Good Generalization**: Doesn't overfit and underfit
- **Prediction**: Perform moderately for Positive Review and ok for Negative Review
- **How to Improve**: Augment more Negative Reviews (0.4-1.0)

Final Best Model





- Test MAE: 0.0758
- Test MSE: 0.0122
- Test RMSE: 0.1106
- Test MSLE:0.0061

Model Insights

- **Performance**: Great (Low Error Metrics)
- **Good Generalization**: Doesn't overfit and underfit
- **Prediction**: Perform great for Positive Review and better for Negative Review