FX Price Change Classification

Prepared by – Aakash Maskara

Overview

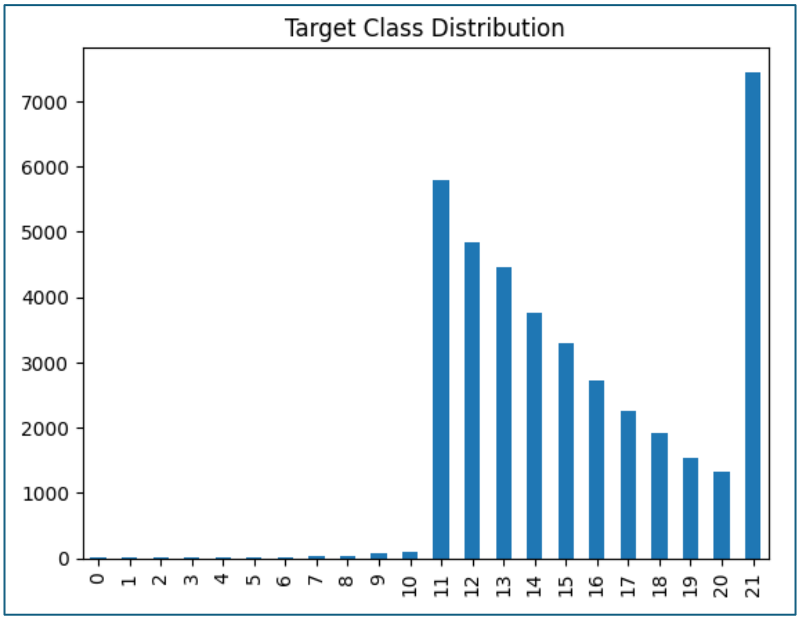
This project builds a supervised learning pipeline using TensorFlow and Keras to predict the fractional change in CAD-high price using historical financial indicators. The target was transformed into a 22-class classification task.

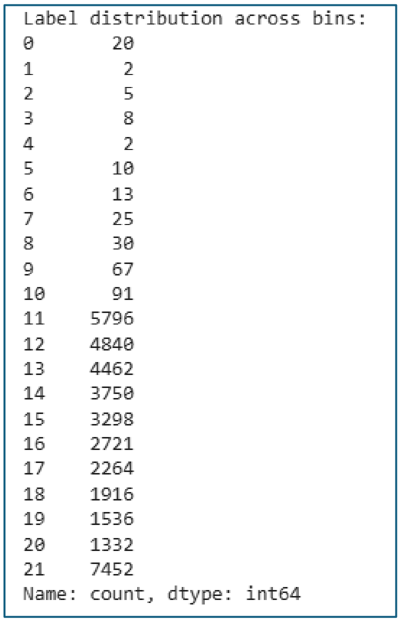
Dataset Insights

- Data was read from appml-assignment1-dataset-v2.pkl, containing historical financial features and timestamps
- The target labels were computed as fractional change between next hour high and current hour close
- This fractional change was digitized into 22 bins using np.digitize

Observation

The dataset is highly skewed, with most samples concentrated between bins 11-21. Bins 0-10 had negligible representation.





```
Label distribution across bins:
0        20
1         2
2         5
3         8
4         2
5        10
6        13
7        25
8        30
9        67
10       91
11     5796
12     4840
13     4462
14     3750
15     3298
16     2721
17     2264
18     1916
19     1536
20     1332
21     7452
Name: count, dtype: int64
```

Preprocessing & Feature Engineering

- Temporal features (weekday, hour, month) were extracted using datetime parsing
- Features were saved in TFRecord format using tf.train.Example
- Missing values in ticker data were handled using a custom ImputeLayer, replacing NaNs with column-wise minimum
- All ticker values were normalized post imputation using Keras Normalization() layer
- Embedding layers were used for the categorical inputs :
  - weekday : 7 categories → Embedding(dim=2)
  - hour : 24 categories → Embedding(dim=2)
  - month : 12 categories → Embedding(dim=2)

Model Architecture

- Input layers for : tickers, weekday, hour, month
- Imputed and normalized tickers passed through a Dense(64,relu) compression layer
- Embedding outputs and ticker features concatenated
- Followed by $2-4$ dense layers with tunable activations and units
- Final Output : Dense(22, softmax) for classification

Training & Optimization

- Hyperparameters tuning with Keras Tuner’s RandomSearch :
  - Parameters tuned : activation, num_layers, unit per layer, learning_rate, batch_size
- Used label smoothing (0.1) to stabilize gradients and mitigate overconfidence
- EarlyStopping callback was used to avoid overfitting

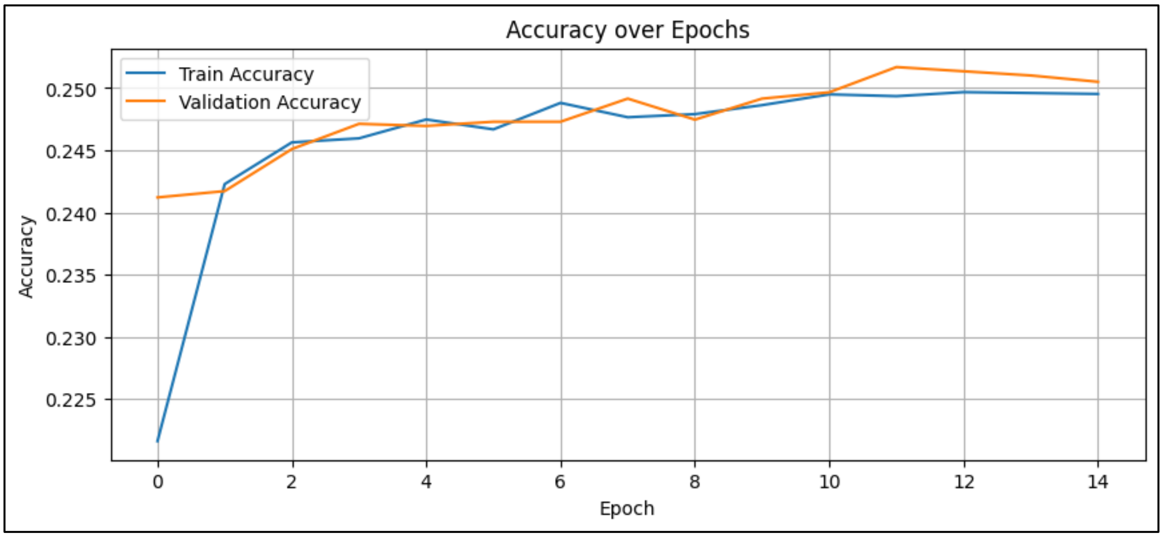
Final Training Results

Accuracy over Epochs

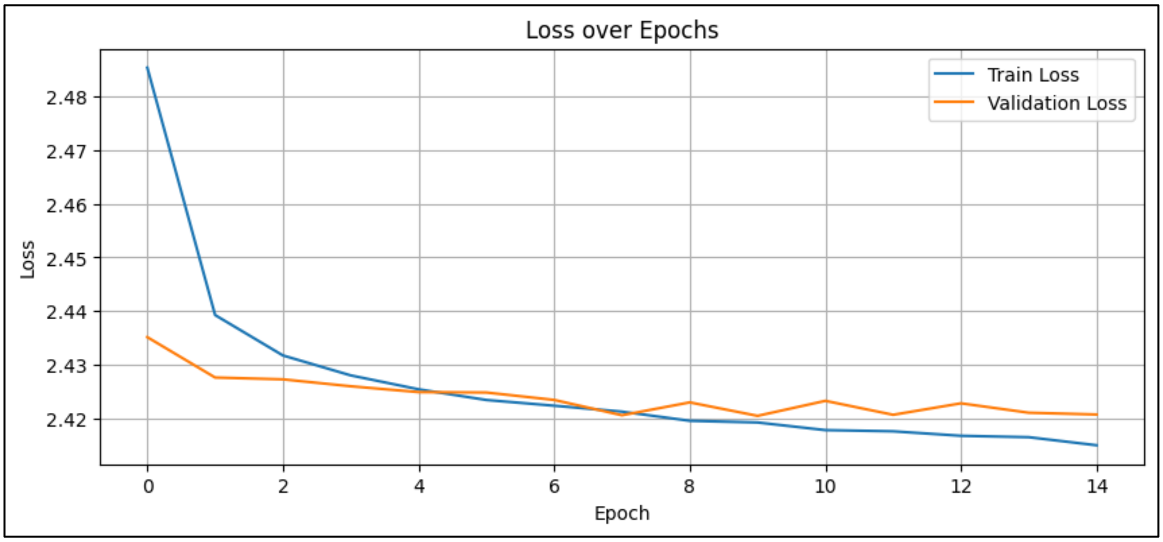
Accuracy observed for Training ~ $22.1\% - 25\%$

Accuracy observed for Validation ~ $24.1\% - 25\%$

Validation and training curves show no overfitting and gradual improvement, validating a well-regularized model.



Loss over Epochs



Final Performance

Test Accuracy = 25%

Model Architecture to achieve this -

- Inputs
  - tickers : 188 – dimensional float vector
  - weekday : int [0-6], embedded to (2,)
  - hour : int [0-23], embedded to (2,)
  - month : int [1-12], embedded to (2,)
- Preprocessing :
  - tickers → ImputedLayer (replaces NaNs with feature-wise min)
  - Imputed tickers → Normalization
  - Compressed via Dense(64, activation=”relu”) before fusion
- Concatenated with embeddings of categorical inputs :
  - weekday, hour, month → Embedding layers + Flatten → Concat
- Hidden Layers :
  - Dense (128, activation = “elu”)
  - Dense (64, activation = “elu”)
  - Dense (128, activation = “elu”)
- Output Layer :
  - Dense (22, activation = “softmax”) for 22-class classification
- Loss Function :
  - Custom Label Smoothing CrossEntropy (smoothing = 0.1)
- Optimizer :
  - Adam with learning rate = 0.001

Experiments Conducted Before Reaching Final Model

Throughout the development of this project, several model variants and techniques were explored to maximize accuracy and generalization performance. Key experiments included:

- GridSearch (Initially) :
  A full grid search was attempted for exhaustive hyperparameter tuning but was replaced with Random Search to reduce computational time while still exploring the hyperparameter space effectively.
- Dropout layers :
  Dropout was introduced in earlier versions to reduce overfitting. However, due to the already sparse representation in certain bins and early convergence behavior, dropout tended to slow learning and was ultimately removed.
- Label smoothing (Final Model) :
  Introduced to prevent overconfident predictions, label smoothing helped stabilize training and improve validation accuracy in a multi-class classification setting.
- Class Imbalance Handling via Class Weights :
  To address severe class imbalance (most labels concentrated in bins 11–21), I experimented with class weighting using sklearn.utils.class_weight.
  However, this approach was scrapped because it caused instability in convergence and did not offer accuracy improvements better than label smoothing.
- Dense layer compression before fusion (Final Model) :
  A Dense (64) compression layer was applied to the ticker input after normalization. This reduced dimensionality and helped in better fusion with the categorical embeddings.
- Embedding Tuning
  Embedding sizes for categorical features (weekday, hour, month) were tuned. A dimension of 2 was finalized as it provided good expressiveness without overfitting
- Exclusion of low-support bins $(0-10)$ (Scrapped) :

  Initially, bins with very low support (0–10) were filtered out to improve class balance. However, to remain compliant with assignment requirements, this was not used in the final model.

Conclusion

Despite class imbalance and non-linearity, the final model generalizes without overfitting, reaching ~25% accuracy on a 22-class problem. The architecture balances embedding, normalization, and robust hyperparameter tuning strategies.