# *Amazon Review Star-Rating Classifier*

In [1]:
```
%matplotlib inline

import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, Dense, GlobalMaxPooling1D, Dr
from tensorflow.keras.models import Sequential
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping

try:
    tf.set_random_seed(1337)                        # set the random seed for re
except:
    tf.random.set_seed(1337)                         # NOTE: Newer version of te
np.random.seed(1337)                            #       instead of tf.set_rando
```

In [3]:
```
import sys
print(sys.executable)
```

```
C:\Users\khanf\anaconda3\python.exe
```

In [ ]:

In [ ]:

In [3]:
```
pip install notebook
```

In [2]:
```
pip install ipykernel
```

In [2]:
```
pip install tensorflow
```

In [ ]:

## Abstract

I built and trained a deep learning model to predict Amazon product review star ratings (1–5) from raw text. Starting with data sampling and preprocessing, I experimented with

network architectures, regularization strategies, and training regimes to maximize accuracy and generalization.

In [5]:
```
!dir "C:\Users\khanf\Downloads\assignment_3"
```

```
 Volume in drive C is Windows-SSD
 Volume Serial Number is C27C-EEEA

 Directory of C:\Users\khanf\Downloads\assignment_3

03/26/2024  11:19 PM    <DIR>          .
03/29/2024  11:50 PM    <DIR>          ..
03/30/2024  03:56 PM    <DIR>          assignment_3
               0 File(s)              0 bytes
               3 Dir(s)  163,428,167,680 bytes free
```

# Setting up and preparing the data

In [6]:
```
amazon_reviews = pd.read_csv('Reviews.csv', nrows=262084)
amazon_reviews.head(5)
```

Out[6]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | Helpfuln |
|---|---|---|---|---|---|---|
| **0** | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | |
| **1** | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | |
| **2** | 3 | B000LQOCH0 | ABXLMWJIXXAIN | Natalia Corres "Natalia Corres" | 1 | |
| **3** | 4 | B000UA0QIQ | A395BORC6FGVXV | Karl | 3 | |
| **4** | 5 | B006K2ZZ7K | A1UQRSCLF8GW1T | Michael D. Bigham "M. Wassir" | 0 | |

In [7]:

```
csv_file_path = r'C:\Users\khanf\Downloads\assignment_3\assignment_3\Reviews

amazon_reviews = pd.read_csv(csv_file_path, nrows=262084)

filtered_reviews = pd.concat([
    amazon_reviews[amazon_reviews['Score'] == i].head(1000) for i in range(1
])

filtered_reviews['Score'] = filtered_reviews['Score'] - 1

X_train, X_test, y_train, y_test = train_test_split(
    filtered_reviews['Text'], filtered_reviews['Score'], test_size=0.2, ranc
)
```

# Tokenizing our texts

I lowercased and removed punctuation, then integer-encoded words and padded sequences. $n$

In [8]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

num_unique_words = len(tokenizer.word_index)

sequences = tokenizer.texts_to_sequences(X_train)

review_lengths = [len(sequence) for sequence in sequences]

percentile_80th = np.percentile(review_lengths, 80)

print(f'Number of unique words: {num_unique_words}')
print(f'80th percentile of review lengths: {percentile_80th}')
```

```
Number of unique words: 13244
80th percentile of review lengths: 119.0
```

---

In [9]:
```
tokenizer = Tokenizer(num_words=20000) #We create the tokenizer using only t
```

In [10]:
```
tokenizer.fit_on_texts(X_train)
```

In [11]:
```
word_index_subset = {k: tokenizer.word_index[k] for k in list(tokenizer.word
print("Word index subset:", word_index_subset)

test_sequence = tokenizer.texts_to_sequences(['I just feel very very good'])
print("Sequence for 'I just feel very very good':", test_sequence)

sequence_to_text = tokenizer.sequences_to_texts([[109, 19, 824, 76, 114, 631
print("Text for the sequence [109, 19, 824, 76, 114, 6315, 1137, 8070]:", se
```

```
Word index subset: {'the': 1, 'i': 2, 'a': 3, 'and': 4, 'to': 5}
Sequence for 'I just feel very very good': [[2, 36, 351, 39, 39, 32]]
Text for the sequence [109, 19, 824, 76, 114, 6315, 1137, 8070]: ['did you m
iss your best data science professor']
```

---

In [12]:
```
train_sequences = tokenizer.texts_to_sequences(X_train)
test_sequences = tokenizer.texts_to_sequences(X_test)gth
train_padded = pad_sequences(train_sequences, maxlen=116, padding='post', tr
test_padded = pad_sequences(test_sequences, maxlen=116, padding='post', trur
print("Shape of train_padded:", train_padded.shape)
print("Shape of test_padded:", test_padded.shape)
```

```
Shape of train_padded: (4000, 116)
Shape of test_padded: (1000, 116)
```

---

In [13]:
```
pip install --upgrade tensorflow
```

## Building a basic neural network model

In [14]:
```
model = Sequential()
model.add(Embedding(20000, 128))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(5, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metr
model.summary()
```

**Model: "sequential"**

| Layer (type) | Output Shape | Par |
|---|---|---|
| embedding (Embedding) | ? | 0 (unbu |
| global_max_pooling1d (GlobalMaxPooling1D) | ? | 0 (unbu |
| dense (Dense) | ? | 0 (unbu |
| dense_1 (Dense) | ? | 0 (unbu |
| dense_2 (Dense) | ? | 0 (unbu |

**Total params:** 0 (0.00 B)

**Trainable params:** 0 (0.00 B)

**Non-trainable params:** 0 (0.00 B)

In [ ]:

In [15]:
```
model.fit(train_padded, y_train, validation_split=0.2, epochs=10)
X_train, X_test, y_train, y_test = train_test_split(
    filtered_reviews['Text'],
    filtered_reviews['Score'],
    test_size=0.2,
    random_state=1337
)
```

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - accuracy: 0.2271 - loss: 1.6077
- val_accuracy: 0.2275 - val_loss: 1.5737
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.3446 - loss: 1.4867
- val_accuracy: 0.4512 - val_loss: 1.2909
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.5455 - loss: 1.0837
- val_accuracy: 0.4625 - val_loss: 1.2287
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.7097 - loss: 0.7262
- val_accuracy: 0.4700 - val_loss: 1.3044
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.8599 - loss: 0.4449
- val_accuracy: 0.4412 - val_loss: 1.7242
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.9156 - loss: 0.2796
- val_accuracy: 0.4500 - val_loss: 2.0449
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.9371 - loss: 0.1947
- val_accuracy: 0.4300 - val_loss: 2.4392
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.9414 - loss: 0.1567
- val_accuracy: 0.4913 - val_loss: 2.1729
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.9758 - loss: 0.0837
- val_accuracy: 0.4938 - val_loss: 2.3740
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9966 - loss: 0.0266
- val_accuracy: 0.4550 - val_loss: 2.6447
```

## Overfitting Mitigation

Analyzing the model's performance over time, we observe signs of overfitting, as indicated by the divergence between training accuracy and validation accuracy. The training accuracy approaches near perfection, which suggests the model may be memorizing the training data rather than learning generalizable patterns. To address overfitting, we could implement regularization techniques. L1 or L2 regularization would penalize the weights of our neural network, encouraging the model to maintain smaller weight values, thus leading to a simpler model less prone to overfitting. Another regularization technique we could apply is dropout, which randomly sets a fraction of input units to zero during training, forcing the network to learn more robust features. Although clustering is typically associated with unsupervised learning, the insights gained from a clustering analysis could inform the feature engineering process, potentially leading to improved model generalization if we can identify and encode meaningful clusters as new features. In essence, the current model's performance indicates that while it has learned to fit the training data well, it lacks the ability to generalize these learnings to unseen data, hence the necessity for regularization strategies to improve its predictive power on new, unseen data.

In [16]:
```
model2 = Sequential()
model2.add(Embedding(input_dim=20000, output_dim=128, input_shape=(116,)))
model2.add(Dropout(0.2))
model2.add(GlobalMaxPooling1D())
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(0.2))
model2.add(Dense(5, activation='softmax'))
model2.compile(loss='sparse_categorical_crossentropy', optimizer='adam', met

model2.fit(train_padded, y_train, validation_split=0.2, epochs=10)
```

```
Epoch 1/10
C:\Users\khanf\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.p
y:81: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
yer. When using Sequential models, prefer using an `Input(shape)` object as
the first layer in the model instead.
  super().__init__(**kwargs)
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 30ms/step - accuracy: 0.2176 - loss: 1.6079
- val_accuracy: 0.2313 - val_loss: 1.5901
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.3000 - loss: 1.5642
- val_accuracy: 0.3425 - val_loss: 1.4895
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.4305 - loss: 1.3939
- val_accuracy: 0.4512 - val_loss: 1.3303
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.5538 - loss: 1.1436
- val_accuracy: 0.4700 - val_loss: 1.2251
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.6725 - loss: 0.9014
- val_accuracy: 0.4888 - val_loss: 1.1909
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.7540 - loss: 0.7065
- val_accuracy: 0.4750 - val_loss: 1.1904
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.8442 - loss: 0.5110
- val_accuracy: 0.4812 - val_loss: 1.2211
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.9135 - loss: 0.3481
- val_accuracy: 0.4800 - val_loss: 1.2968
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.9571 - loss: 0.2255
- val_accuracy: 0.4650 - val_loss: 1.3631
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.9742 - loss: 0.1537
- val_accuracy: 0.4712 - val_loss: 1.4132
```

Out[16]:
```
<keras.src.callbacks.history.History at 0x253cdb8ce10>
```

## Exercise 4:

Modify the neural network definition above to try and fix the overfitting problem using Dropout. Explain the configuration that you tried and your results. Why do you think your modifications were or were not able to mitigate the overfitting problem?

In [17]:
```
model2.add(Dropout(0.5))

from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=3)
model2.fit(train_padded, y_train, validation_split=0.2, epochs=10, callbacks
```

```
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.9835 - loss: 0.1052
- val_accuracy: 0.4812 - val_loss: 1.4730
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.9854 - loss: 0.0777
- val_accuracy: 0.4875 - val_loss: 1.5249
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.9977 - loss: 0.0457
- val_accuracy: 0.4650 - val_loss: 1.5754
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.9983 - loss: 0.0332
- val_accuracy: 0.4700 - val_loss: 1.6223
```

Out[17]:
```
<keras.src.callbacks.history.History at 0x253d13355d0>
```

After applying dropout regularization and observing the training process, it became evident that the model continues to overfit. Despite the high accuracy on the training set, the validation accuracy does not follow suit and the validation loss even increases slightly with each epoch. This discrepancy suggests that the model's predictions are not generalizing well to the validation set, reaffirming the presence of overfitting. Future steps could include introducing L1/L2 regularization, applying early stopping, reducing the complexity of the neural network, or increasing the dropout rate. Additionally, tweaking the learning rate or augmenting the training data might also improve the model's generalization capabilities. It's crucial to find a balance between the model's ability to learn from the training data and its capacity to apply these learnings to new data effectively.

## 8. Early Stopping & Checkpointing

L2 regularization on all Dense layers

In [18]:
```
from tensorflow.keras.regularizers import l2

model = Sequential()
model.add(Embedding(20000, 128))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dense(5, activation='softmax', kernel_regularizer=l2(0.01)))
```

Rationale: L2 regularization penalizes the square values of the weights, which discourages large weights more severely than small weights. Applying L2 regularization on all Dense layers encourages the network to spread out the importance of features, potentially improving generalization. Expected Outcome: Improved validation accuracy without a significant increase in validation loss, indicating reduced overfitting.

L1 regularization on the first Dense layer

In [19]:
```
from tensorflow.keras.regularizers import l1

model = Sequential()
model.add(Embedding(20000, 128))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu', kernel_regularizer=l1(0.01)))
model.add(Dense(128, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

Rationale: L1 regularization penalizes the absolute value of the weights, leading to a sparse model where some weights can become exactly zero. This could be beneficial for feature selection if some features are irrelevant. Expected Outcome: A more sparse model with potentially improved interpretability, but the effectiveness in reducing overfitting may vary.

Combination of L1 and L2 regularization (L1L2) on the last Dense layer

In [20]:
```
from tensorflow.keras.regularizers import l1_l2

model = Sequential()
model.add(Embedding(20000, 128))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(5, activation='softmax', kernel_regularizer=l1_l2(l1=0.01, l
```

Rationale: Combining L1 and L2 regularizations brings together the benefits of both: sparsity from L1 and weight penalty from L2. Applying it to the output layer could help make the final decision-making process more robust to noise in the features. Expected Outcome: A balanced approach that might lead to a model that is both sparse and generalizes better to unseen data.

L2 regularization on the Embedding layer

In [21]:
```
model = Sequential()
model.add(Embedding(20000, 128, embeddings_regularizer=l2(0.01)))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

Rationale: Regularizing the Embedding layer can help in controlling the magnitude of the embedding vectors, potentially leading to more general representations. Expected

Outcome: Subtle improvements in generalization, as the embeddings are regularized to not overfit to specific words or contexts in the training set.

Increasing regularization strength progressively in Dense layers

In [22]:
```
model = Sequential()
model.add(Embedding(20000, 128))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dense(5, activation='softmax', kernel_regularizer=l2(0.1)))
```

Rationale: Gradually increasing the regularization strength encourages earlier layers to learn more general features, while allowing deeper layers to fine-tune the decision-making process with a stronger constraint against overfitting. Expected Outcome: A model that balances learning complex patterns and maintaining generalizability, possibly showing the best improvement against overfitting among the configurations.

# Regularization through adding more data

In [25]:
```
def create_and_train_model(X_train, y_train, X_val, y_val, embedding_dim=128
    model = Sequential([
        Embedding(input_dim=20000, output_dim=embedding_dim),
        GlobalMaxPooling1D(),
        Dense(dense_units, activation='relu'),
        Dropout(dropout_rate),
        Dense(dense_units, activation='relu'),
        Dense(5, activation='softmax')
    ])

    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',

    history = model.fit(
        X_train, y_train,
        epochs=10,
        validation_data=(X_val, y_val)
    )

    return model, history

def preprocess_text(df, num_words=20000, maxlen=116):
    tokenizer = Tokenizer(num_words=num_words)
    tokenizer.fit_on_texts(df['Text'])

    sequences = tokenizer.texts_to_sequences(df['Text'])
    padded_sequences = pad_sequences(sequences, maxlen=maxlen)

    return padded_sequences, df['Score'] - 1

df = pd.read_csv('Reviews.csv', usecols=['Text', 'Score'])
```

```
filtered_reviews = pd.concat([df[df['Score'] == i].head(1000) for i in range
X, y = preprocess_text(filtered_reviews)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, rando

model_original, history_original = create_and_train_model(X_train, y_train,

filtered_reviews_larger = pd.concat([df[df['Score'] == i].head(6000) for i i
X_large, y_large = preprocess_text(filtered_reviews_larger)
X_train_large, X_val_large, y_train_large, y_val_large = train_test_split(X_

model_larger, history_larger = create_and_train_model(X_train_large, y_train
```

```
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 28ms/step - accuracy: 0.2134 - loss: 1.6085
- val_accuracy: 0.2800 - val_loss: 1.5480
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.3639 - loss: 1.4558
- val_accuracy: 0.4550 - val_loss: 1.2781
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.5394 - loss: 1.1022
- val_accuracy: 0.4890 - val_loss: 1.2364
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.6744 - loss: 0.8160
- val_accuracy: 0.4930 - val_loss: 1.3336
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.7875 - loss: 0.5596
- val_accuracy: 0.4930 - val_loss: 1.6007
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.8690 - loss: 0.3861
- val_accuracy: 0.4970 - val_loss: 1.7211
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.8899 - loss: 0.3005
- val_accuracy: 0.4290 - val_loss: 2.2315
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9060 - loss: 0.2638
- val_accuracy: 0.4740 - val_loss: 2.3445
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.9355 - loss: 0.1753
- val_accuracy: 0.4950 - val_loss: 2.3678
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9601 - loss: 0.1149
- val_accuracy: 0.4890 - val_loss: 2.6232
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 22s 27ms/step - accuracy: 0.3081 - loss: 1.4753
- val_accuracy: 0.4993 - val_loss: 1.1710
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 19s 26ms/step - accuracy: 0.5173 - loss: 1.1162
- val_accuracy: 0.5198 - val_loss: 1.1386
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 19s 26ms/step - accuracy: 0.6144 - loss: 0.9388
- val_accuracy: 0.5195 - val_loss: 1.2411
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.6993 - loss: 0.7603
- val_accuracy: 0.5152 - val_loss: 1.3893
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.7705 - loss: 0.6019
- val_accuracy: 0.5202 - val_loss: 1.6771
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 20s 27ms/step - accuracy: 0.8295 - loss: 0.4637
- val_accuracy: 0.5123 - val_loss: 1.7997
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.8498 - loss: 0.3989
- val_accuracy: 0.5042 - val_loss: 1.9358
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.8699 - loss: 0.3403
- val_accuracy: 0.5017 - val_loss: 2.0833
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.8885 - loss: 0.2889
```

```
- val_accuracy: 0.5122 - val_loss: 2.1303
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.9078 - loss: 0.2367
- val_accuracy: 0.5143 - val_loss: 2.5856
```

To test the hypothesis that adding more data would result in a more generalizable model, we trained the original neural network model with an increased dataset size, expanding from 1,000 to 6,000 reviews for each score category. The comparison between the original model and the model trained on a larger dataset reveals that the latter achieves slightly higher and more stable validation accuracy, alongside a more controlled increase in validation loss. This indicates improved model generalization capabilities, likely due to the diversity and volume of data allowing the model to learn more robust features. Despite these improvements, signs of overfitting are still present, suggesting that while adding more data can enhance model performance, additional strategies may be necessary to fully address overfitting.

# Regularization through early stopping

In [28]:

```python
from tensorflow.keras.callbacks import EarlyStopping
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    verbose=1,
    restore_best_weights=True
)

history = model.fit(
    train_padded, y_train,
    validation_split=0.2,
    epochs=50,
    callbacks=[early_stopping]
)
```

```
Epoch 1/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 27ms/step - accuracy: 0.1905 - loss: 3.2216
- val_accuracy: 0.2087 - val_loss: 1.9663
Epoch 2/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.1867 - loss: 1.8439
- val_accuracy: 0.2087 - val_loss: 1.6534
Epoch 3/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.1851 - loss: 1.6378
- val_accuracy: 0.1988 - val_loss: 1.6142
Epoch 4/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.2075 - loss: 1.6126
- val_accuracy: 0.1988 - val_loss: 1.6099
Epoch 5/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.2075 - loss: 1.6098
- val_accuracy: 0.1988 - val_loss: 1.6095
Epoch 6/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.2075 - loss: 1.6095
- val_accuracy: 0.1988 - val_loss: 1.6094
Epoch 7/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.2075 - loss: 1.6095
- val_accuracy: 0.1988 - val_loss: 1.6094
Epoch 8/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.2075 - loss: 1.6095
- val_accuracy: 0.1988 - val_loss: 1.6094
Epoch 9/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.2075 - loss: 1.6095
- val_accuracy: 0.1988 - val_loss: 1.6094
Epoch 10/50
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.2075 - loss: 1.6095
- val_accuracy: 0.1988 - val_loss: 1.6094
Epoch 10: early stopping
Restoring model weights from the end of the best epoch: 7.
```

# Evaluating our model

Unlike in most previous cases, we used *three*Unlike most of my earlier projects, I split the data into three sets—training, validation, and test—instead of just two. I've done all of my model tuning on the validation set and haven't touched the test set since I carved it off at the very beginning. In my experiments, it's essential that I run the model on the test set only once. Because there's so much randomness in play, I must avoid cherry-picking the best results. I'm free to optimize and iterate on the validation set as much as I need, but I'll keep the test set locked away until the end and report all final metrics from that single, pre-determined evaluation.

## Error Analysis

In [37]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GlobalMaxPooling1D, Dense, Dr
from tensorflow.keras.callbacks import EarlyStopping

X_train, X_test, y_train, y_test = train_test_split(
    filtered_reviews['Text'].astype(str),
    filtered_reviews['Score'],
    test_size=0.2,
    random_state=42
)
tokenizer = Tokenizer(num_words=20000)
tokenizer.fit_on_texts(X_train)
X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_test_sequences = tokenizer.texts_to_sequences(X_test)

maxlen = 116
X_train_padded = pad_sequences(X_train_sequences, maxlen=maxlen)
X_test_padded = pad_sequences(X_test_sequences, maxlen=maxlen)

model = Sequential([
    Input(shape=(maxlen,)),
    Embedding(20000, 128),
    GlobalMaxPooling1D(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(5, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metr

early_stopping = EarlyStopping(monitor='val_loss', patience=3, verbose=1, re

model.fit(X_train_padded, y_train, validation_split=0.2, epochs=50, callback

test_loss, test_acc = model.evaluate(X_test_padded, y_test, verbose=2)
print(f'Test Accuracy: {test_acc}, Test Loss: {test_loss}')

predictions = model.predict(X_test_padded)
predicted_classes = np.argmax(predictions, axis=1)

misclassified_idx = np.where(predicted_classes != y_test.to_numpy())[0]
print(f'Found {len(misclassified_idx)} misclassified examples.')

for idx in misclassified_idx[:5]:
    print(f'Review (misclassified): {X_test.iloc[idx]}')
    print(f'Actual Label: {y_test.iloc[idx]}, Predicted Label: {predicted_cl
```

```
Epoch 1/50
100/100 - 4s - 37ms/step - accuracy: 0.2072 - loss: 1.6083 - val_accuracy:
0.3300 - val_loss: 1.5951
Epoch 2/50
100/100 - 2s - 24ms/step - accuracy: 0.3047 - loss: 1.5729 - val_accuracy:
0.4100 - val_loss: 1.5260
Epoch 3/50
100/100 - 2s - 25ms/step - accuracy: 0.4028 - loss: 1.4390 - val_accuracy:
0.4487 - val_loss: 1.3676
Epoch 4/50
100/100 - 2s - 24ms/step - accuracy: 0.5191 - loss: 1.2131 - val_accuracy:
0.4638 - val_loss: 1.2514
Epoch 5/50
100/100 - 2s - 25ms/step - accuracy: 0.6334 - loss: 0.9836 - val_accuracy:
0.4675 - val_loss: 1.2008
Epoch 6/50
100/100 - 2s - 25ms/step - accuracy: 0.7428 - loss: 0.7591 - val_accuracy:
0.4725 - val_loss: 1.1927
Epoch 7/50
100/100 - 2s - 25ms/step - accuracy: 0.8350 - loss: 0.5631 - val_accuracy:
0.4663 - val_loss: 1.2276
Epoch 8/50
100/100 - 2s - 24ms/step - accuracy: 0.9022 - loss: 0.3904 - val_accuracy:
0.4837 - val_loss: 1.2956
Epoch 9/50
100/100 - 2s - 25ms/step - accuracy: 0.9506 - loss: 0.2582 - val_accuracy:
0.4775 - val_loss: 1.3696
Epoch 9: early stopping
Restoring model weights from the end of the best epoch: 6.
32/32 - 0s - 9ms/step - accuracy: 0.5020 - loss: 1.2077
Test Accuracy: 0.5019999742507935, Test Loss: 1.2077020406723022
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Found 498 misclassified examples.
Review (misclassified): My golden retriever loves these bones!  I ordered th
em and they arrived in a timely manner.  However, when I opened the box, the
y were old - meaning the filling was somewhat dried and didn't completely fi
ll the hoof due to shrinkage.  Once the plastic is removed, the filling pret
ty much falls out.  Very disappointing!
Actual Label: 2, Predicted Label: 1

Review (misclassified): I like the fact that this vanilla flavor has no alco
hol in it so i could add it to my baby's pancakes.  It does not taste the sa
me to me as the regular stuff.  It is rather bland in my opinion.
Actual Label: 1, Predicted Label: 2

Review (misclassified): These little taffy's are expensive. I think I paid a
round $11 for the pound. They taste good, but man do they give me gas. After
eating about 3 my stomach would be rumbling for a few hours and I would have
to pass gas a lot. My girlfriend wouldn't let me eat them before bed hahaha.
There was never any pain involved, just gas. I know it was these doing it be
cause we tested it over a few different days. I wouldn't eat the taffy some
days and would be fine. All it took was a few pieces and an hour late the bu
bble guts would start.<br /><br />This information may be useful since its n
ot on amazon. Sugar free does not = carb free. A 1 pound bag is 453g.<br /><
br />From the manufacturer website:<br />ingredients: hydrogenated starch hy
drolysate, partially hydrogenated soybean and/or coconut oil, acid combinati
```

```
ons for flavor (citric acid and/or tartaric acid), egg albumin, salt, lecith
in, sucralose, natural and artificial flavors, combinations of red 40, blue
1, blue 2, yellow 5, yellow6.<br /><br />Nutrition Facts<br />Serving Size
(40g)<br /><br />Amount Per Serving<br />Calories  110  Cal from fat 20<br /
>  % Daily Value<br />Total Fat 2.5g  4%<br />Saturated Fat 0g  0%<br />Tran
s Fat 0g<br />Cholesterol 0mg  0%<br />Sodium  25mg  1%<br />Total Carbohydr
ate 28g  9%<br />Sugar 0g<br />Protein 0g
Actual Label: 0, Predicted Label: 3

Review (misclassified): It was hard to grow but once it grew my cat loved i
t. I haven't found any seeds that work well yet, but I am still looking. I w
ould buy this again is=f I had better luck with growing it.
Actual Label: 2, Predicted Label: 1

Review (misclassified): They're ok I guess but each bag has about a pound of
salt in it. You'll find yourself brushing away salt from each piece just for
it to taste somewhat decent. Also, no matter what you set the timer on the m
icrowave, you will have little burn/hardened pieces. So you're not even gett
ing a whole lot out of one bag.
Actual Label: 1, Predicted Label: 2
```

After training my model with early stopping and evaluating it on the test set, I achieved a test accuracy of about 50.2% and a test loss of 1.2077. This tells me that, while my model correctly predicts the star rating for roughly half of the reviews, there's still plenty of room to improve. When I looked at the misclassified examples, it became clear that the model struggles with subtle, mixed sentiments. For instance, one review praised the product at first but then complained it was stale, and another lauded its non-alcoholic flavor for a baby's food yet called it bland. Those nuanced, "both positive and negative" opinions confused the classifier, highlighting its current inability to fully grasp complex emotional cues in text. To address this, I'm considering several strategies. Increasing the dataset size could give the model a richer variety of sentiment examples. Moving to more advanced architectures—like LSTMs or Transformers—would help it understand context and nuance more deeply. I could also engineer additional features (for example, sentiment scores or part-of-speech tags) to give the model clearer signals about tone and emphasis. In conclusion, this work lays a solid foundation for text-review classification, but it's just a first step. I plan to experiment with richer data, more sophisticated models, and targeted feature engineering to boost accuracy and sensitivity to linguistic subtleties. Ultimately, this reminds me that there's no one-size-fits-all solution: neural networks often require extensive experimentation, and sometimes the quality and quantity of data matter even more than the model architecture itself.

In [ ]: