```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
import seaborn as sns
import matplotlib.pyplot as plt

file_path = r'C:\Users\V Varunkumar\Desktop\Assignments\Data mining\DM 3\Grocery_Items_31.csv'
data = pd.read_csv(file_path)
all_items = data.apply(pd.Series.explode).stack()
unique_items = all_items.unique()
item_counts = all_items.value_counts()
num_unique_items = len(unique_items)
num_records = len(data)
most_popular_item = item_counts.idxmax()
most_popular_count = item_counts.max()
print(f"Number of unique items: {num_unique_items}")
print(f"Number of records (transactions): {num_records}")
print(f"Most popular item: {most_popular_item}")
print(f"Number of transactions containing the most popular item: {most_popular_count}")
```

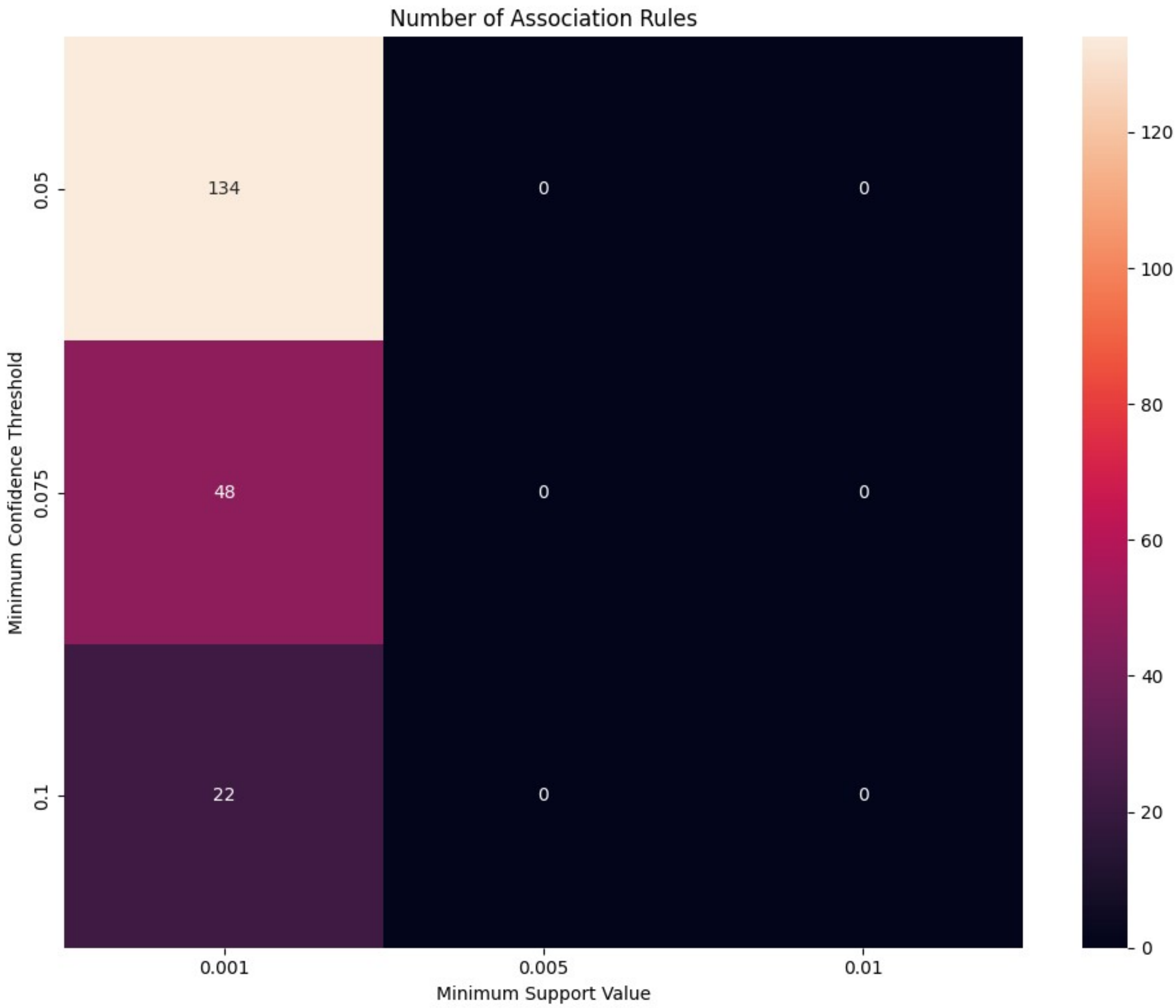
```
Number of unique items: 166
Number of records (transactions): 8000
Most popular item: whole milk
Number of transactions containing the most popular item: 1309
```

```python
transactions = data.apply(lambda x: x.dropna().tolist(), axis=1).tolist()
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_data = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_data, columns=te.columns_)
from mlxtend.frequent_patterns import apriori
frequent_itemsets = apriori(df_encoded, min_support=0.01, use_colnames=True)
from mlxtend.frequent_patterns import association_rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.08)
print("Association Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

```
Association Rules:
          antecedents         consequents   support  confidence
lift
0  (other vegetables)        (rolls/buns)  0.010750    0.084729
0.763325
```

```
1         (rolls/buns)  (other vegetables)  0.010750    0.096847
0.763325
2         (whole milk)  (other vegetables)  0.014750    0.096091
0.757369
3  (other vegetables)        (whole milk)  0.014750    0.116256
0.757369
4         (whole milk)        (rolls/buns)  0.015125    0.098534
0.887696
5         (rolls/buns)        (whole milk)  0.015125    0.136261
0.887696
6               (soda)        (whole milk)  0.010375    0.110226
0.718083
7             (yogurt)        (whole milk)  0.011000    0.127168
0.828454
```

```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, association_rules
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore", message=".*backend2gui.*")
data_encoded = pd.get_dummies(data)
msv_values = [0.001, 0.005, 0.01]
mct_values = [0.05, 0.075, 0.1]
rule_counts = np.zeros((len(mct_values), len(msv_values)))

for i, msv in enumerate(msv_values):
    frequent_itemsets = apriori(data_encoded, min_support=msv,
use_colnames=True)
    for j, mct in enumerate(mct_values):
        rules = association_rules(frequent_itemsets,
metric="confidence", min_threshold=mct)
        rule_counts[j, i] = len(rules)

plt.figure(figsize=(10, 8))
sns.heatmap(rule_counts, annot=True, xticklabels=msv_values,
yticklabels=mct_values, fmt='g')
plt.xlabel('Minimum Support Value')
plt.ylabel('Minimum Confidence Threshold')
plt.title('Number of Association Rules')
plt.tight_layout()
plt.show()
```




```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, 
Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator


img_width, img_height = 64, 64
batch_size = 32
num_classes = 4
epochs = 20

dataset_dir = r'C:\Users\V Varunkumar\Desktop\Assignments\Data mining\
DM 3\images'
datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    validation_split=0.2
```

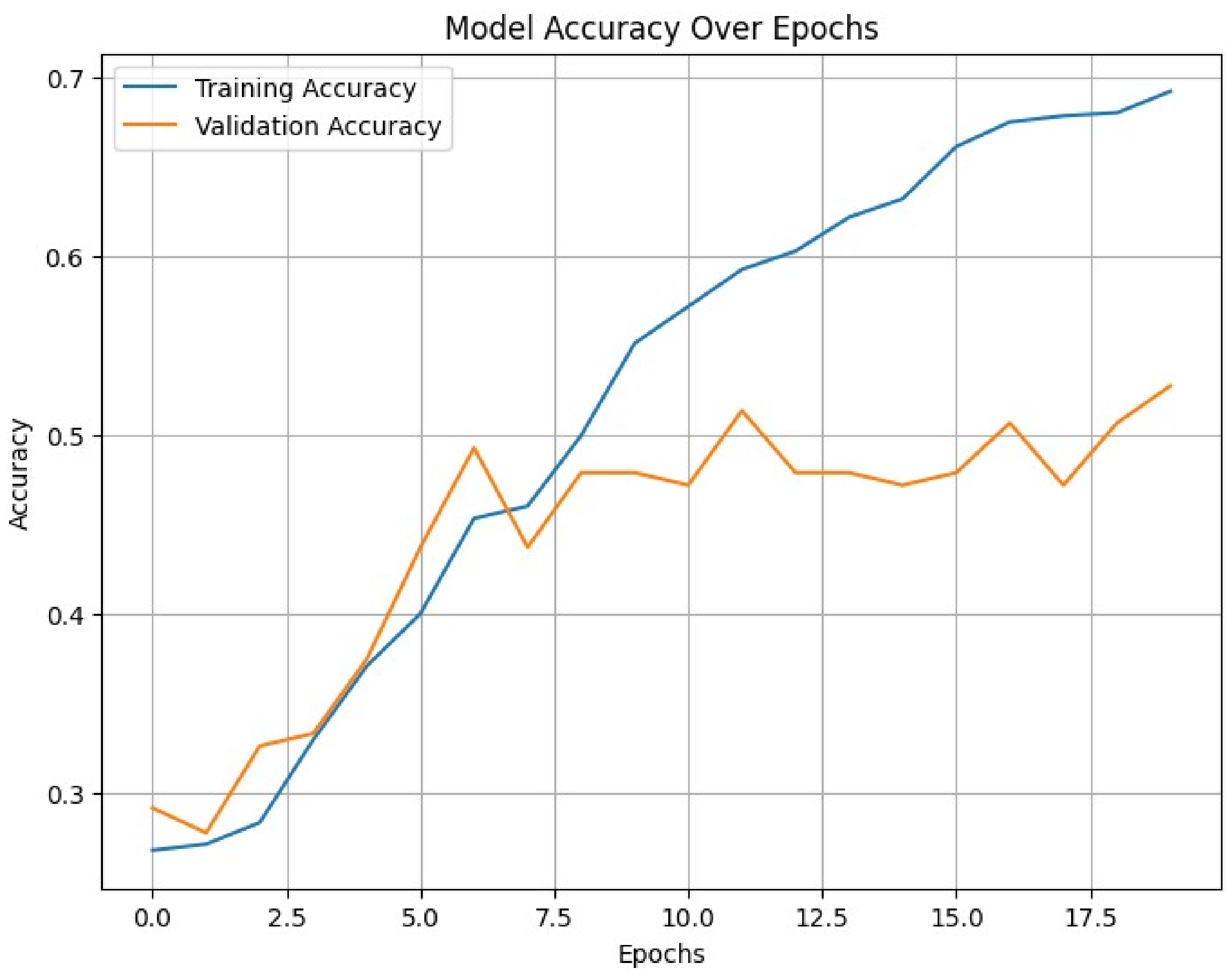
```
)
train_generator = datagen.flow_from_directory(
    dataset_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training',
    shuffle=True
)
validation_generator = datagen.flow_from_directory(
    dataset_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=False
)
model = Sequential([
    Conv2D(8, (3, 3), activation='relu', input_shape=(img_width,
img_height, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(4, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(8, activation='relu'),
    Dense(num_classes, activation='softmax')
])
model.compile(
    optimizer=Adam(),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator
)
val_loss, val_accuracy = model.evaluate(validation_generator)
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy:
{val_accuracy:.4f}")
```

```
Found 582 images belonging to 4 classes.
Found 144 images belonging to 4 classes.
Epoch 1/20
19/19 [==============================] - 4s 142ms/step - loss: 1.3913
- accuracy: 0.2680 - val_loss: 1.3804 - val_accuracy: 0.2917
Epoch 2/20
19/19 [==============================] - 2s 109ms/step - loss: 1.3766
- accuracy: 0.2715 - val_loss: 1.3721 - val_accuracy: 0.2778
```

```
Epoch 3/20
19/19 [==============================] - 2s 110ms/step - loss: 1.3659
- accuracy: 0.2835 - val_loss: 1.3569 - val_accuracy: 0.3264
Epoch 4/20
19/19 [==============================] - 2s 105ms/step - loss: 1.3443
- accuracy: 0.3299 - val_loss: 1.3316 - val_accuracy: 0.3333
Epoch 5/20
19/19 [==============================] - 2s 115ms/step - loss: 1.3091
- accuracy: 0.3711 - val_loss: 1.2927 - val_accuracy: 0.3750
Epoch 6/20
19/19 [==============================] - 2s 115ms/step - loss: 1.2764
- accuracy: 0.4003 - val_loss: 1.2673 - val_accuracy: 0.4375
Epoch 7/20
19/19 [==============================] - 2s 108ms/step - loss: 1.2087
- accuracy: 0.4536 - val_loss: 1.1860 - val_accuracy: 0.4931
Epoch 8/20
19/19 [==============================] - 2s 108ms/step - loss: 1.1744
- accuracy: 0.4605 - val_loss: 1.2155 - val_accuracy: 0.4375
Epoch 9/20
19/19 [==============================] - 2s 118ms/step - loss: 1.1202
- accuracy: 0.5000 - val_loss: 1.1767 - val_accuracy: 0.4792
Epoch 10/20
19/19 [==============================] - 2s 114ms/step - loss: 1.0615
- accuracy: 0.5515 - val_loss: 1.1679 - val_accuracy: 0.4792
Epoch 11/20
19/19 [==============================] - 2s 111ms/step - loss: 1.0255
- accuracy: 0.5722 - val_loss: 1.1783 - val_accuracy: 0.4722
Epoch 12/20
19/19 [==============================] - 2s 116ms/step - loss: 0.9973
- accuracy: 0.5928 - val_loss: 1.1862 - val_accuracy: 0.5139
Epoch 13/20
19/19 [==============================] - 2s 109ms/step - loss: 0.9582
- accuracy: 0.6031 - val_loss: 1.2115 - val_accuracy: 0.4792
Epoch 14/20
19/19 [==============================] - 2s 117ms/step - loss: 0.9354
- accuracy: 0.6220 - val_loss: 1.2550 - val_accuracy: 0.4792
Epoch 15/20
19/19 [==============================] - 2s 116ms/step - loss: 0.9279
- accuracy: 0.6323 - val_loss: 1.1981 - val_accuracy: 0.4722
Epoch 16/20
19/19 [==============================] - 2s 114ms/step - loss: 0.8774
- accuracy: 0.6615 - val_loss: 1.2172 - val_accuracy: 0.4792
Epoch 17/20
19/19 [==============================] - 2s 127ms/step - loss: 0.8513
- accuracy: 0.6753 - val_loss: 1.2195 - val_accuracy: 0.5069
Epoch 18/20
19/19 [==============================] - 2s 108ms/step - loss: 0.8336
- accuracy: 0.6787 - val_loss: 1.2151 - val_accuracy: 0.4722
Epoch 19/20
```

```
19/19 [==============================] - 2s 100ms/step - loss: 0.8021
- accuracy: 0.6804 - val_loss: 1.2270 - val_accuracy: 0.5069
Epoch 20/20
19/19 [==============================] - 2s 113ms/step - loss: 0.7727
- accuracy: 0.6924 - val_loss: 1.2347 - val_accuracy: 0.5278
5/5 [==============================] - 1s 106ms/step - loss: 1.2347 -
accuracy: 0.5278
Validation Loss: 1.2347, Validation Accuracy: 0.5278
```

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



```
#Banner ID 916501290
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
img_width, img_height = 64, 64
batch_size = 32
num_classes = 4
epochs = 20
datagen = ImageDataGenerator(rescale=1.0 / 255.0,
validation_split=0.2)

train_generator = datagen.flow_from_directory(
    dataset_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training',
    shuffle=True
)

validation_generator = datagen.flow_from_directory(
    dataset_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=False
)

def train_cnn_with_filter(filter_size, epochs=20):
    print(f"Training CNN with filter size {filter_size} ×
{filter_size} in the second convolutional layer...")
    model = Sequential([
        Conv2D(8, (3, 3), activation='relu', input_shape=(img_width,
img_height, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(4, (filter_size, filter_size), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(8, activation='relu'),
        Dense(num_classes, activation='softmax')
    ])


    model.compile(
        optimizer=Adam(),
```

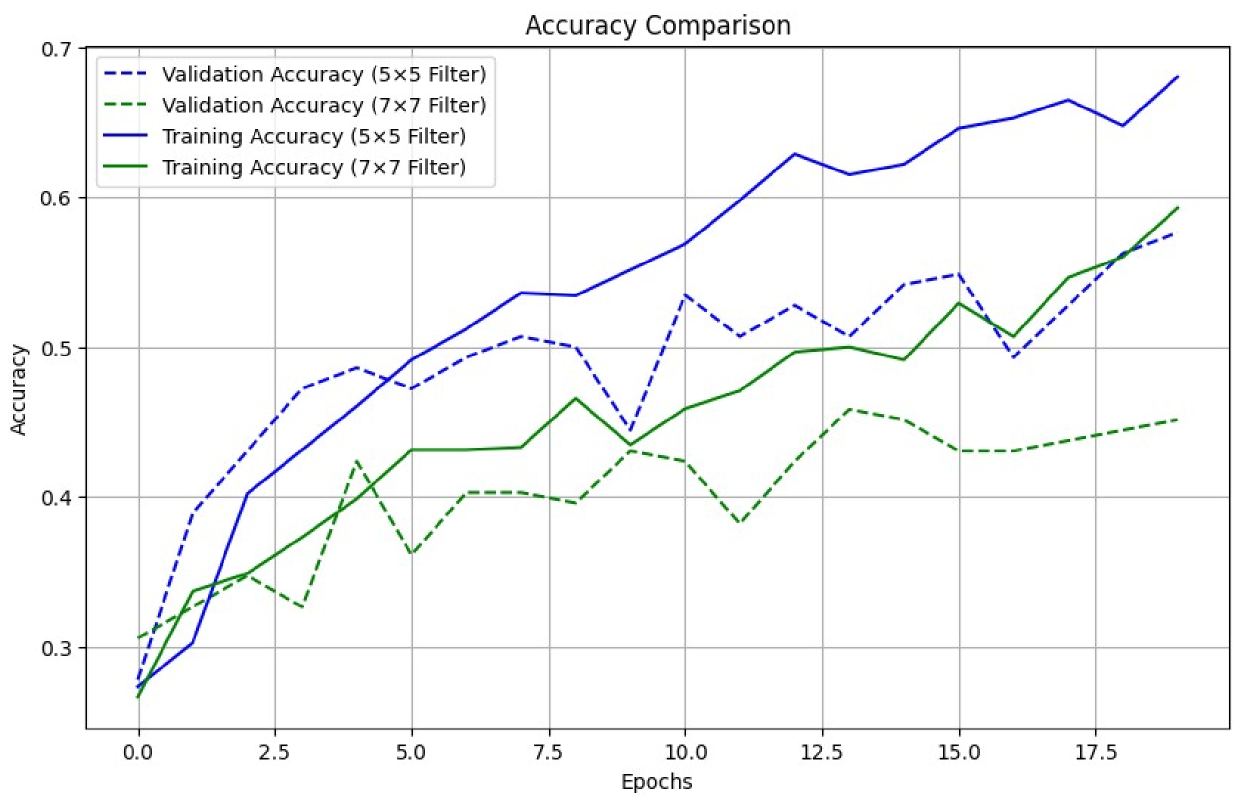
```
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )


    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=validation_generator
    )


    val_loss, val_accuracy = model.evaluate(validation_generator)
    print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy:
{val_accuracy:.4f}")

    return history, model

history_5x5, model_5x5 = train_cnn_with_filter(filter_size=5,
epochs=epochs)


history_7x7, model_7x7 = train_cnn_with_filter(filter_size=7,
epochs=epochs)

plt.figure(figsize=(10, 6))
plt.plot(history_5x5.history['val_accuracy'], label='Validation
Accuracy (5×5 Filter)', linestyle='--', color='blue')
plt.plot(history_7x7.history['val_accuracy'], label='Validation
Accuracy (7×7 Filter)', linestyle='--', color='green')
plt.plot(history_5x5.history['accuracy'], label='Training Accuracy
(5×5 Filter)', color='blue')
plt.plot(history_7x7.history['accuracy'], label='Training Accuracy
(7×7 Filter)', color='green')
plt.title('Accuracy Comparison')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

```
Found 582 images belonging to 4 classes.
Found 144 images belonging to 4 classes.
Training CNN with filter size 5 × 5 in the second convolutional
layer...
Epoch 1/20
19/19 [==============================] - 6s 211ms/step - loss: 1.3775
- accuracy: 0.2732 - val_loss: 1.3737 - val_accuracy: 0.2778
Epoch 2/20
19/19 [==============================] - 2s 129ms/step - loss: 1.3524
```

```
- accuracy: 0.3024 - val_loss: 1.3524 - val_accuracy: 0.3889
Epoch 3/20
19/19 [==============================] - 4s 193ms/step - loss: 1.3195
- accuracy: 0.4021 - val_loss: 1.3135 - val_accuracy: 0.4306
Epoch 4/20
19/19 [==============================] - 4s 198ms/step - loss: 1.2754
- accuracy: 0.4313 - val_loss: 1.2863 - val_accuracy: 0.4722
Epoch 5/20
19/19 [==============================] - 3s 171ms/step - loss: 1.2313
- accuracy: 0.4605 - val_loss: 1.2352 - val_accuracy: 0.4861
Epoch 6/20
19/19 [==============================] - 3s 187ms/step - loss: 1.1652
- accuracy: 0.4914 - val_loss: 1.2404 - val_accuracy: 0.4722
Epoch 7/20
19/19 [==============================] - 3s 162ms/step - loss: 1.1072
- accuracy: 0.5120 - val_loss: 1.1743 - val_accuracy: 0.4931
Epoch 8/20
19/19 [==============================] - 4s 204ms/step - loss: 1.0798
- accuracy: 0.5361 - val_loss: 1.1787 - val_accuracy: 0.5069
Epoch 9/20
19/19 [==============================] - 4s 202ms/step - loss: 1.0530
- accuracy: 0.5344 - val_loss: 1.1416 - val_accuracy: 0.5000
Epoch 10/20
19/19 [==============================] - 4s 184ms/step - loss: 1.0120
- accuracy: 0.5515 - val_loss: 1.1830 - val_accuracy: 0.4444
Epoch 11/20
19/19 [==============================] - 4s 219ms/step - loss: 1.0151
- accuracy: 0.5687 - val_loss: 1.1700 - val_accuracy: 0.5347
Epoch 12/20
19/19 [==============================] - 2s 125ms/step - loss: 0.9602
- accuracy: 0.5979 - val_loss: 1.1595 - val_accuracy: 0.5069
Epoch 13/20
19/19 [==============================] - 3s 142ms/step - loss: 0.9259
- accuracy: 0.6289 - val_loss: 1.1168 - val_accuracy: 0.5278
Epoch 14/20
19/19 [==============================] - 2s 118ms/step - loss: 0.9190
- accuracy: 0.6151 - val_loss: 1.0999 - val_accuracy: 0.5069
Epoch 15/20
19/19 [==============================] - 2s 112ms/step - loss: 0.8780
- accuracy: 0.6220 - val_loss: 1.0935 - val_accuracy: 0.5417
Epoch 16/20
19/19 [==============================] - 2s 112ms/step - loss: 0.8562
- accuracy: 0.6460 - val_loss: 1.0841 - val_accuracy: 0.5486
Epoch 17/20
19/19 [==============================] - 2s 129ms/step - loss: 0.8390
- accuracy: 0.6529 - val_loss: 1.1199 - val_accuracy: 0.4931
Epoch 18/20
19/19 [==============================] - 3s 135ms/step - loss: 0.8435
- accuracy: 0.6649 - val_loss: 1.1435 - val_accuracy: 0.5278
```

```
Epoch 19/20
19/19 [==============================] - 2s 128ms/step - loss: 0.8468
- accuracy: 0.6478 - val_loss: 1.1157 - val_accuracy: 0.5625
Epoch 20/20
19/19 [==============================] - 2s 128ms/step - loss: 0.8077
- accuracy: 0.6804 - val_loss: 1.1385 - val_accuracy: 0.5764
5/5 [==============================] - 1s 89ms/step - loss: 1.1385 -
accuracy: 0.5764
Validation Loss: 1.1385, Validation Accuracy: 0.5764
Training CNN with filter size 7 × 7 in the second convolutional
layer...
Epoch 1/20
19/19 [==============================] - 4s 142ms/step - loss: 1.3894
- accuracy: 0.2663 - val_loss: 1.3679 - val_accuracy: 0.3056
Epoch 2/20
19/19 [==============================] - 2s 121ms/step - loss: 1.3621
- accuracy: 0.3368 - val_loss: 1.3548 - val_accuracy: 0.3264
Epoch 3/20
19/19 [==============================] - 2s 123ms/step - loss: 1.3233
- accuracy: 0.3488 - val_loss: 1.3438 - val_accuracy: 0.3472
Epoch 4/20
19/19 [==============================] - 2s 111ms/step - loss: 1.3025
- accuracy: 0.3729 - val_loss: 1.3193 - val_accuracy: 0.3264
Epoch 5/20
19/19 [==============================] - 2s 128ms/step - loss: 1.2706
- accuracy: 0.3986 - val_loss: 1.3001 - val_accuracy: 0.4236
Epoch 6/20
19/19 [==============================] - 2s 127ms/step - loss: 1.2574
- accuracy: 0.4313 - val_loss: 1.2847 - val_accuracy: 0.3611
Epoch 7/20
19/19 [==============================] - 2s 124ms/step - loss: 1.2456
- accuracy: 0.4313 - val_loss: 1.2758 - val_accuracy: 0.4028
Epoch 8/20
19/19 [==============================] - 2s 114ms/step - loss: 1.2086
- accuracy: 0.4330 - val_loss: 1.2518 - val_accuracy: 0.4028
Epoch 9/20
19/19 [==============================] - 2s 116ms/step - loss: 1.1882
- accuracy: 0.4656 - val_loss: 1.2481 - val_accuracy: 0.3958
Epoch 10/20
19/19 [==============================] - 2s 118ms/step - loss: 1.1740
- accuracy: 0.4347 - val_loss: 1.2614 - val_accuracy: 0.4306
Epoch 11/20
19/19 [==============================] - 2s 115ms/step - loss: 1.1504
- accuracy: 0.4588 - val_loss: 1.2417 - val_accuracy: 0.4236
Epoch 12/20
19/19 [==============================] - 2s 124ms/step - loss: 1.1350
- accuracy: 0.4708 - val_loss: 1.2518 - val_accuracy: 0.3819
Epoch 13/20
19/19 [==============================] - 2s 128ms/step - loss: 1.1181
```

```
- accuracy: 0.4966 - val_loss: 1.2739 - val_accuracy: 0.4236
Epoch 14/20
19/19 [==============================] - 2s 130ms/step - loss: 1.1137
- accuracy: 0.5000 - val_loss: 1.2816 - val_accuracy: 0.4583
Epoch 15/20
19/19 [==============================] - 2s 131ms/step - loss: 1.1061
- accuracy: 0.4914 - val_loss: 1.2175 - val_accuracy: 0.4514
Epoch 16/20
19/19 [==============================] - 2s 130ms/step - loss: 1.0571
- accuracy: 0.5292 - val_loss: 1.2884 - val_accuracy: 0.4306
Epoch 17/20
19/19 [==============================] - 2s 124ms/step - loss: 1.0601
- accuracy: 0.5069 - val_loss: 1.2421 - val_accuracy: 0.4306
Epoch 18/20
19/19 [==============================] - 3s 133ms/step - loss: 1.0312
- accuracy: 0.5464 - val_loss: 1.2726 - val_accuracy: 0.4375
Epoch 19/20
19/19 [==============================] - 2s 115ms/step - loss: 0.9993
- accuracy: 0.5601 - val_loss: 1.2885 - val_accuracy: 0.4444
Epoch 20/20
19/19 [==============================] - 2s 130ms/step - loss: 0.9612
- accuracy: 0.5928 - val_loss: 1.2746 - val_accuracy: 0.4514
5/5 [==============================] - 0s 79ms/step - loss: 1.2746 -
accuracy: 0.4514
Validation Loss: 1.2746, Validation Accuracy: 0.4514
```



Describe and discuss what you observe by comparing the performance of the first model and the other two models you constructed in (a), (b) or (c) (depending on which one you did). Comment on whether the models are overfit, underfit, or just right.

First Model: It is showing signs of overfitting with high training accuracy (~70%) but lower and fluctuating validation accuracy (~50%).

5×5 Filter Model: It Performs better than the 7×7 model but exhibits overfitting as the training accuracy (~60%) significantly outpaces validation accuracy (~55%).

7×7 Filter Model: It shows underfit with lower training accuracy (~55%) and closely matching validation accuracy, indicating it may be too simple to capture the data patterns.

```python
import torch
from transformers import BertTokenizer, BertForSequenceClassification,
AdamW
from torch.utils.data import DataLoader, TensorDataset
import json
import numpy as np
from sklearn.metrics import accuracy_score, f1_score
import matplotlib.pyplot as plt
```

```
WARNING:tensorflow:From c:\Users\V Varunkumar\AppData\Local\Programs\
Python\Python311\Lib\site-packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

```python
import json
import torch
from torch.utils.data import TensorDataset, DataLoader
from transformers import BertTokenizer

def load_json_file(filepath):
    with open(filepath, 'r', encoding='utf-8') as f:
        data = [json.loads(line.strip()) for line in f]
    return data

train_data = load_json_file(r'C:\Users\V Varunkumar\Desktop\
Assignments\Data mining\DM 3\student_31\train.json')
test_data = load_json_file(r'C:\Users\V Varunkumar\Desktop\
Assignments\Data mining\DM 3\student_31\test.json')
val_data = load_json_file(r'C:\Users\V Varunkumar\Desktop\Assignments\
Data mining\DM 3\student_31\validation.json')

print("Structure of first item in train_data:")
print(json.dumps(train_data[0], indent=2))

all_labels = ['anger', 'anticipation', 'disgust', 'fear', 'joy',
              'love', 'optimism', 'pessimism', 'sadness', 'surprise',
'trust']

def convert_labels_to_list(data, label_classes):
    print("Keys in data item:", list(data[0].keys()))
    for item in data:
        item['labels'] = [float(item[label]) for label in
label_classes]
    return data

train_data = convert_labels_to_list(train_data, all_labels)
val_data = convert_labels_to_list(val_data, all_labels)
test_data = convert_labels_to_list(test_data, all_labels)
```

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

def encode_texts(data):
    text_key = 'Tweet'
    if text_key not in data[0]:
        raise KeyError(f"'{text_key}' not found in data. Available
keys: {list(data[0].keys())}")
    texts = [item[text_key] for item in data]
    return tokenizer(texts, padding=True, truncation=True,
max_length=128, return_tensors='pt')

try:
    train_encodings = encode_texts(train_data)
    test_encodings = encode_texts(test_data)
    val_encodings = encode_texts(val_data)
    print("Encoding successful")
except Exception as e:
    print(f"Error during encoding: {e}")
    raise

train_labels = torch.tensor([item['labels'] for item in train_data],
dtype=torch.float)
test_labels = torch.tensor([item['labels'] for item in test_data],
dtype=torch.float)
val_labels = torch.tensor([item['labels'] for item in val_data],
dtype=torch.float)

print("Train labels shape:", train_labels.shape)
print("Test labels shape:", test_labels.shape)
print("Validation labels shape:", val_labels.shape)
print("Train encodings shape:", train_encodings['input_ids'].shape)
print("Test encodings shape:", test_encodings['input_ids'].shape)
print("Validation encodings shape:", val_encodings['input_ids'].shape)

def create_dataloader(encodings, labels, batch_size=16):
    input_ids = encodings['input_ids']
    attention_mask = encodings['attention_mask']
    dataset = TensorDataset(input_ids, attention_mask, labels)
    return DataLoader(dataset, batch_size=batch_size, shuffle=True)

try:
    train_dataloader = create_dataloader(train_encodings,
train_labels)
    val_dataloader = create_dataloader(val_encodings, val_labels)
    test_dataloader = create_dataloader(test_encodings, test_labels)

    print("Train dataloader size:", len(train_dataloader))
    print("Validation dataloader size:", len(val_dataloader))
    print("Test dataloader size:", len(test_dataloader))
except Exception as e:
```

```
    print(f"Error creating dataloaders: {e}")
    print("Shape of train_encodings:", {k: v.shape for k, v in
train_encodings.items()})
    print("Shape of train_labels:", train_labels.shape)
```

```
Structure of first item in train_data:
{
  "ID": "2017-En-31264",
  "Tweet": "And the weathers so breezy, man why can't life always be
this easy?",
  "anger": false,
  "anticipation": false,
  "disgust": false,
  "fear": false,
  "joy": true,
  "love": false,
  "optimism": true,
  "pessimism": false,
  "sadness": false,
  "surprise": true,
  "trust": false
}
Keys in data item: ['ID', 'Tweet', 'anger', 'anticipation', 'disgust',
'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadness', 'surprise',
'trust']
Keys in data item: ['ID', 'Tweet', 'anger', 'anticipation', 'disgust',
'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadness', 'surprise',
'trust']
Keys in data item: ['ID', 'Tweet', 'anger', 'anticipation', 'disgust',
'fear', 'joy', 'love', 'optimism', 'pessimism', 'sadness', 'surprise',
'trust']
Encoding successful
Train labels shape: torch.Size([3000, 11])
Test labels shape: torch.Size([1500, 11])
Validation labels shape: torch.Size([400, 11])
Train encodings shape: torch.Size([3000, 63])
Test encodings shape: torch.Size([1500, 57])
Validation encodings shape: torch.Size([400, 65])
Train dataloader size: 188
Validation dataloader size: 25
Test dataloader size: 94
```

```
from transformers import BertForSequenceClassification
from torch.optim import AdamW

num_labels = len(all_labels)

model = BertForSequenceClassification.from_pretrained('bert-base-
uncased',
```

```
num_labels=num_labels,

problem_type="multi_label_classification")

optimizer = AdamW(model.parameters(), lr=2e-5)

print(f"Model initialized with {num_labels} output labels")
print(f"Optimizer initialized with learning rate 2e-5")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print(f"Model moved to {device}")
```

```
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at bert-base-uncased and are newly
initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```
Model initialized with 11 output labels
Optimizer initialized with learning rate 2e-5
Model moved to cpu
```

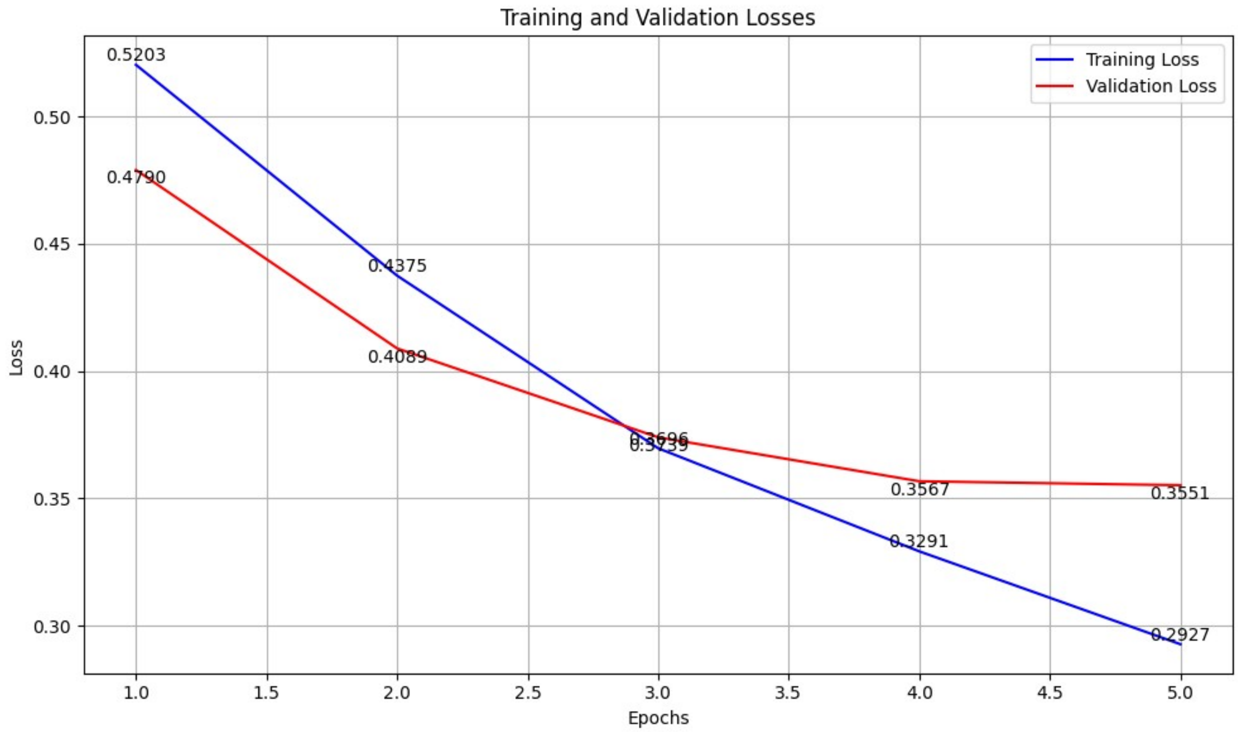
```
import json
import torch
from torch.utils.data import TensorDataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification
from torch.optim import AdamW
from tqdm import tqdm

def load_json_file(filepath, max_samples=None):
    with open(filepath, 'r', encoding='utf-8') as f:
        data = [json.loads(line.strip()) for line in f]
    if max_samples:
        return data[:max_samples]
    return data

max_samples = 1000
train_data = load_json_file(r'C:\Users\V Varunkumar\Desktop\
Assignments\Data mining\DM 3\student_31\train.json', max_samples)
test_data = load_json_file(r'C:\Users\V Varunkumar\Desktop\
Assignments\Data mining\DM 3\student_31\test.json', max_samples//5)
val_data = load_json_file(r'C:\Users\V Varunkumar\Desktop\Assignments\
Data mining\DM 3\student_31\validation.json', max_samples//5)

print(f"Loaded {len(train_data)} train samples, {len(val_data)}
validation samples, {len(test_data)} test samples")

all_labels = ['anger', 'anticipation', 'disgust', 'fear', 'joy',
              'love', 'optimism', 'pessimism', 'sadness', 'surprise',
'trust']
```

```
def convert_labels_to_list(data, label_classes):
    for item in data:
        item['labels'] = [float(item[label]) for label in
label_classes]
    return data

train_data = convert_labels_to_list(train_data, all_labels)
val_data = convert_labels_to_list(val_data, all_labels)
test_data = convert_labels_to_list(test_data, all_labels)


tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

def encode_texts(data):
    text_key = 'Tweet'
    texts = [item[text_key] for item in data]
    return tokenizer(texts, padding=True, truncation=True,
max_length=128, return_tensors='pt')

train_encodings = encode_texts(train_data)
val_encodings = encode_texts(val_data)
test_encodings = encode_texts(test_data)

train_labels = torch.tensor([item['labels'] for item in train_data],
dtype=torch.float)
val_labels = torch.tensor([item['labels'] for item in val_data],
dtype=torch.float)
test_labels = torch.tensor([item['labels'] for item in test_data],
dtype=torch.float)

def create_dataloader(encodings, labels, batch_size=16):
    dataset = TensorDataset(encodings['input_ids'],
encodings['attention_mask'], labels)
    return DataLoader(dataset, batch_size=batch_size, shuffle=True)

train_dataloader = create_dataloader(train_encodings, train_labels)
val_dataloader = create_dataloader(val_encodings, val_labels)
test_dataloader = create_dataloader(test_encodings, test_labels)

num_labels = len(all_labels)
model = BertForSequenceClassification.from_pretrained('bert-base-
uncased',

num_labels=num_labels,

problem_type="multi_label_classification")

optimizer = AdamW(model.parameters(), lr=2e-5)
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

num_epochs = 5
train_losses = []
val_losses = []

for epoch in range(num_epochs):
    model.train()
    total_train_loss = 0
    progress_bar = tqdm(train_dataloader, desc=f'Epoch
{epoch+1}/{num_epochs} [Train]')

    for batch in progress_bar:
        input_ids, attention_mask, labels = [b.to(device) for b in
batch]

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask,
labels=labels)
        loss = outputs.loss
        total_train_loss += loss.item()

        loss.backward()
        optimizer.step()

        progress_bar.set_postfix({'train_loss': f'{loss.item():.4f}'})

    avg_train_loss = total_train_loss / len(train_dataloader)
    train_losses.append(avg_train_loss)

    model.eval()
    total_val_loss = 0
    with torch.no_grad():
        for batch in tqdm(val_dataloader, desc=f'Epoch
{epoch+1}/{num_epochs} [Val]'):
            input_ids, attention_mask, labels = [b.to(device) for b in
batch]
            outputs = model(input_ids, attention_mask=attention_mask,
labels=labels)
            total_val_loss += outputs.loss.item()

    avg_val_loss = total_val_loss / len(val_dataloader)
    val_losses.append(avg_val_loss)

    print(f'Epoch {epoch+1}/{num_epochs}:')
    print(f'Train Loss: {avg_train_loss:.4f}')
    print(f'Validation Loss: {avg_val_loss:.4f}')
    print('-' * 50)
```

```
print("Training completed!")

# Save the model
torch.save(model.state_dict(), 'bert_multi_label_model.pth')
print("Model saved!")

Loaded 1000 train samples, 200 validation samples, 200 test samples

Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at bert-base-uncased and are newly
initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
Epoch 1/5 [Train]: 100%|██████████| 63/63 [07:38<00:00,  7.28s/it,
train_loss=0.4195]
Epoch 1/5 [Val]: 100%|██████████| 13/13 [00:19<00:00,  1.47s/it]

Epoch 1/5:
Train Loss: 0.5203
Validation Loss: 0.4790
--------------------------------------------------

Epoch 2/5 [Train]: 100%|██████████| 63/63 [06:32<00:00,  6.23s/it,
train_loss=0.4702]
Epoch 2/5 [Val]: 100%|██████████| 13/13 [00:20<00:00,  1.58s/it]

Epoch 2/5:
Train Loss: 0.4375
Validation Loss: 0.4089
--------------------------------------------------

Epoch 3/5 [Train]: 100%|██████████| 63/63 [06:27<00:00,  6.15s/it,
train_loss=0.3825]
Epoch 3/5 [Val]: 100%|██████████| 13/13 [00:29<00:00,  2.24s/it]

Epoch 3/5:
Train Loss: 0.3696
Validation Loss: 0.3739
--------------------------------------------------

Epoch 4/5 [Train]: 100%|██████████| 63/63 [06:14<00:00,  5.95s/it,
train_loss=0.3074]
Epoch 4/5 [Val]: 100%|██████████| 13/13 [00:18<00:00,  1.45s/it]

Epoch 4/5:
Train Loss: 0.3291
Validation Loss: 0.3567
--------------------------------------------------

Epoch 5/5 [Train]: 100%|██████████| 63/63 [06:10<00:00,  5.88s/it,
train_loss=0.2550]
Epoch 5/5 [Val]: 100%|██████████| 13/13 [00:18<00:00,  1.44s/it]
```

```
Epoch 5/5:
Train Loss: 0.2927
Validation Loss: 0.3551
--------------------------------------------------
Training completed!
Model saved!

import matplotlib.pyplot as plt

def plot_learning_curves(train_losses, val_losses):
    epochs = range(1, len(train_losses) + 1)

    plt.figure(figsize=(10, 6))
    plt.plot(epochs, train_losses, 'b-', label='Training Loss')
    plt.plot(epochs, val_losses, 'r-', label='Validation Loss')
    plt.title('Training and Validation Losses')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    # Add value labels
    for i, (train_loss, val_loss) in enumerate(zip(train_losses, 
val_losses)):
        plt.text(i+1, train_loss, f'{train_loss:.4f}', ha='center', 
va='bottom')
        plt.text(i+1, val_loss, f'{val_loss:.4f}', ha='center', 
va='top')

    plt.tight_layout()
    plt.savefig('learning_curves.png')
    plt.show()

# Plot the learning curves
plot_learning_curves(train_losses, val_losses)
```


Training and Validation Losses
Training Loss
Validation Loss
Loss
Epochs
0.5203
0.4790
0.4375
0.4089
0.3696
0.3739
0.3567
0.3551
0.3291
0.2927
0.50
0.45
0.40
0.35
0.30
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0