1)Association Rule Generation from Transaction Data

(a) Download transaction dataset to your local drive.

(b) Download the 'Grocery Items {DATASET NUMBER}.csv' file from the Google Drive Link.DATASET NUMBER is the number assigned to you earlier in the semester.

```
import pandas as pd

# Load the dataset
file_path = r'D:\Data Mining\Programming Assignment - 3\Data Files\
Grocery_Items_5.csv'
data = pd.read_csv(file_path)

# Flatten the dataset to count unique items and their occurrences
flattened_items = data.values.flatten()

# Remove NaN entries
flattened_items = [item for item in flattened_items if
pd.notnull(item)]
```

(c) • How many unique items are there in your dataset? • How many records are there in your dataset? • What is the most popular item in your dataset? How many transactions contain this item?

```
# Count unique items and their occurrences
unique_items = set(flattened_items)
item_counts = pd.Series(flattened_items).value_counts()

# Most popular item and its transaction count
most_popular_item = item_counts.idxmax()
most_popular_count = item_counts.max()

unique_item_count = len(unique_items)
total_records = len(data)

print(f"Number of Unique Items: {unique_item_count}")
print(f"Number of Records (Transactions): {total_records}")
print(f"Most Popular Item: {most_popular_item}")
print(f"Transactions Containing '{most_popular_item}':
{most_popular_count}")

Number of Unique Items: 166
Number of Records (Transactions): 8000
Most Popular Item: whole milk
Transactions Containing 'whole milk': 1354
```

(d) Using minimum support = 0.01 and minimum confidence threshold = 0.08, what are the association rules you can extract from your dataset?

```python
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
import pandas as pd

# Convert the cleaned dataset into a list of transactions, excluding
empty items
transactions = data.fillna("").astype(str).values.tolist()
transactions = [[item for item in transaction if item] for transaction
in transactions]

# Prepare the dataset for apriori algorithm
te = TransactionEncoder()
transformed_data = te.fit(transactions).transform(transactions)
df = pd.DataFrame(transformed_data, columns=te.columns_)

# Remove any empty string column if it exists
if "" in df.columns:
    df = df.drop("", axis=1)

# Generate frequent itemsets with a minimum support of 0.01
frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)

# Generate association rules with a minimum confidence of 0.08
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.08)

print(rules)
```

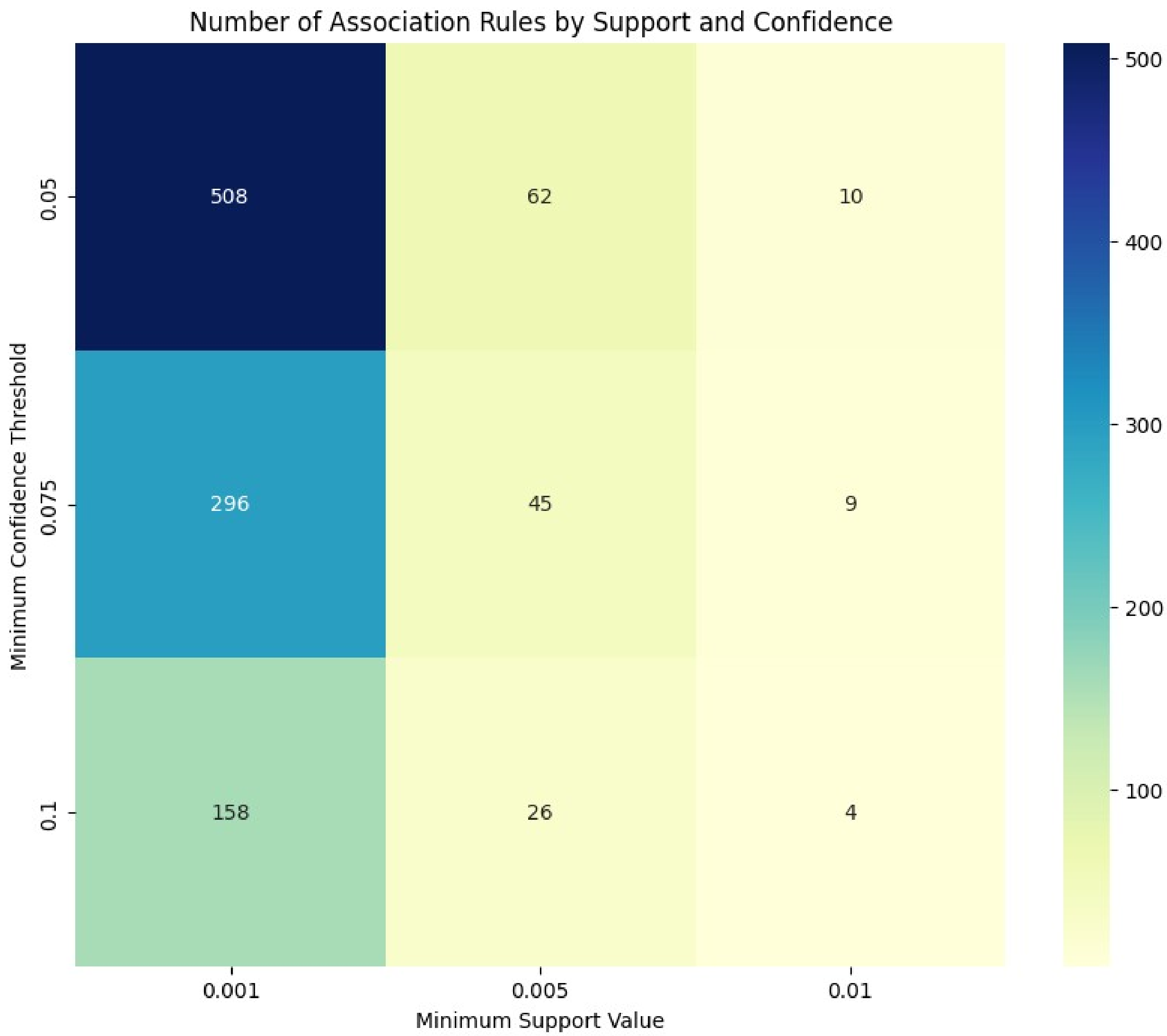
```
          antecedents          consequents  antecedent support  \
0  (other vegetables)         (rolls/buns)            0.122625
1        (rolls/buns)  (other vegetables)            0.108875
2  (other vegetables)         (whole milk)            0.122625
3        (whole milk)  (other vegetables)            0.158375
4        (whole milk)         (rolls/buns)            0.158375
5        (rolls/buns)         (whole milk)            0.108875
6              (soda)         (whole milk)            0.098250
7            (yogurt)         (whole milk)            0.084000

   consequent support   support  confidence      lift  leverage
conviction
0            0.108875  0.010250    0.083588  0.767744 -0.003101
0.972407
1            0.122625  0.010250    0.094145  0.767744 -0.003101
0.968560
2            0.158375  0.014000    0.114169  0.720879 -0.005421
0.950097
3            0.122625  0.014000    0.088398  0.720879 -0.005421
0.962454
4            0.108875  0.013125    0.082873  0.761175 -0.004118
0.971648
```

```
5             0.158375  0.013125    0.120551  0.761175 -0.004118
0.956991
6             0.158375  0.012125    0.123410  0.779224 -0.003435
0.960112
7             0.158375  0.010500    0.125000  0.789266 -0.002803
0.961857
```

(e) Use minimum support values (msv): 0.001, 0.005, 0.01 and minimum confidence threshold (mct): 0.05, 0.075, 0.1. For each pair (msv, mct), find the number of association rules extracted from the dataset. Construct a heatmap using Seaborn data visualization library (https://seaborn.pydata.org/generated/seaborn.heatmap.html) to show the count results such that the x-axis is msv and the y-axis is mct.

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


# Prepare transactions
transactions = data.fillna("").astype(str).values.tolist()
transactions = [[item for item in transaction if item] for transaction
in transactions]

# Encode the transactions
te = TransactionEncoder()
transformed_data = te.fit(transactions).transform(transactions)
df = pd.DataFrame(transformed_data, columns=te.columns_)
if "" in df.columns:
    df = df.drop("", axis=1)

# Define combinations of msv and mct
support_values = [0.001, 0.005, 0.01]
confidence_values = [0.05, 0.075, 0.1]
results = []

# Run apriori and extract rules for each combination
for support in support_values:
    frequent_itemsets = apriori(df, min_support=support,
use_colnames=True)
    for confidence in confidence_values:
        rules = association_rules(frequent_itemsets,
metric="confidence", min_threshold=confidence)
        results.append({'msv': support, 'mct': confidence,
'rules_count': len(rules)})

# Convert results to DataFrame
results_df = pd.DataFrame(results)
```

```python
result_pivot = results_df.pivot(index='mct', columns='msv', values='rules_count')

# Create heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(result_pivot, annot=True, cmap="YlGnBu", fmt="d")
plt.title('Number of Association Rules by Support and Confidence')
plt.xlabel('Minimum Support Value')
plt.ylabel('Minimum Confidence Threshold')
plt.show()
```



2)Image Classification using CNN Construct a 4-class classification model using a convolutional neural network with the following simple architecture (2 point) i 1 Convolutional Layer with 8 3 × 3 filters. ii 1 max pooling with 2 × 2 pool size i 1 Convolutional Layer with 4 3 × 3 filters. ii 1 max pooling with 2 × 2 pool size iii Flatten the Tensor iv 1 hidden layer with 8 nodes for fully connected neural network v Output layer has 4 nodes (since 4 classes) using 'softmax' activation function. (Use 'Relu' for all layers except the output layer.) for 20 epochs using 'adam' optimizer and 'categorical cross entropy' loss function. If your machine is too slow, you can reduce to 5 epochs. You can perform more epochs (> 20) if you want to. For validation split, you will use 20%. For batch size, you can pick a size that will not slow down the training process on your machine. (see https://keras.io/examples/vision/mnist_convnet/)

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Path to your dataset
data_dir = 'D:\\Data Mining\\Programming Assignment - 1\\Data Files\\
Images'

# Image Data Generator with a validation split
train_datagen = ImageDataGenerator(rescale=1./255,
validation_split=0.2)

# Training data generator
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='training')

# Validation data generator
validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='validation')

# Model architecture
model = Sequential([
    Input(shape=(150, 150, 3)),
    Conv2D(8, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(4, (3, 3), activation='relu'),
```

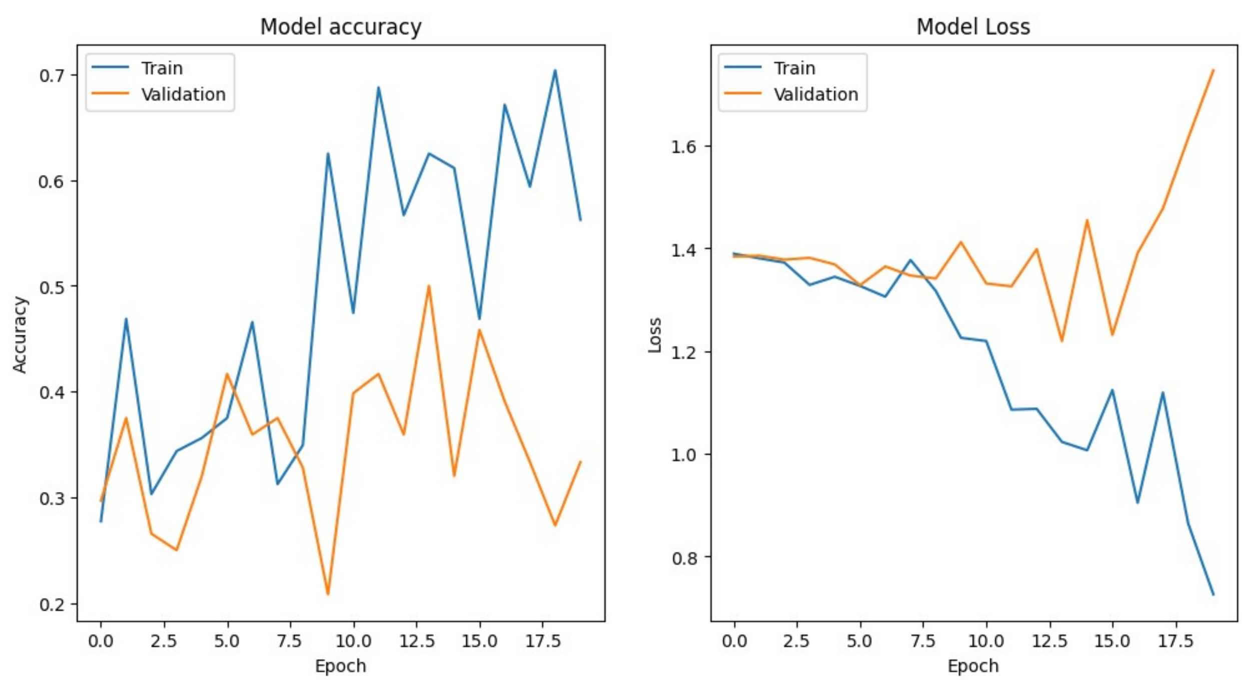
```
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(8, activation='relu'),
    Dense(4, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples //
train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples //
validation_generator.batch_size,
    epochs=20)

# Plot training & validation accuracy values
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Found 616 images belonging to 4 classes.
Found 152 images belonging to 4 classes.
```

```
D:\Data Mining\Programming Assignment - 1\venv\Lib\site-packages\
keras\src\trainers\data_adapters\py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call
`super().__init__(**kwargs)` in its constructor. `**kwargs` can
include `workers`, `use_multiprocessing`, `max_queue_size`. Do not
```

```
pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()

Epoch 1/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 6s 261ms/step - accuracy: 0.2571 - loss: 
1.3937 - val_accuracy: 0.2969 - val_loss: 1.3839
Epoch 2/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.4688 - loss: 
1.3806 - val_accuracy: 0.3750 - val_loss: 1.3858
Epoch 3/20

C:\Users\deepu\AppData\Local\Programs\Python\Python312\Lib\
contextlib.py:158: UserWarning: Your input ran out of data; 
interrupting training. Make sure that your dataset or generator can 
generate at least `steps_per_epoch * epochs` batches. You may need to 
use the `.repeat()` function when building your dataset.
  self.gen.throw(value)

19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.3083 - loss: 
1.3762 - val_accuracy: 0.2656 - val_loss: 1.3781
Epoch 4/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3438 - loss: 
1.3289 - val_accuracy: 0.2500 - val_loss: 1.3815
Epoch 5/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step - accuracy: 0.3208 - loss: 
1.3483 - val_accuracy: 0.3203 - val_loss: 1.3688
Epoch 6/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3750 - loss: 
1.3268 - val_accuracy: 0.4167 - val_loss: 1.3284
Epoch 7/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 71ms/step - accuracy: 0.4645 - loss: 
1.3015 - val_accuracy: 0.3594 - val_loss: 1.3648
Epoch 8/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3125 - loss: 
1.3774 - val_accuracy: 0.3750 - val_loss: 1.3471
Epoch 9/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.3407 - loss: 
1.3288 - val_accuracy: 0.3281 - val_loss: 1.3414
Epoch 10/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6250 - loss: 
1.2258 - val_accuracy: 0.2083 - val_loss: 1.4122
Epoch 11/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.4786 - loss: 
1.2295 - val_accuracy: 0.3984 - val_loss: 1.3317
Epoch 12/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6875 - loss: 
1.0860 - val_accuracy: 0.4167 - val_loss: 1.3261
Epoch 13/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.5931 - loss: 
1.0616 - val_accuracy: 0.3594 - val_loss: 1.3986
```

```
Epoch 14/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6250 - loss:
1.0232 - val_accuracy: 0.5000 - val_loss: 1.2196
Epoch 15/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 75ms/step - accuracy: 0.5918 - loss:
1.0522 - val_accuracy: 0.3203 - val_loss: 1.4548
Epoch 16/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4688 - loss:
1.1242 - val_accuracy: 0.4583 - val_loss: 1.2315
Epoch 17/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.6889 - loss:
0.8970 - val_accuracy: 0.3906 - val_loss: 1.3911
Epoch 18/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5938 - loss:
1.1193 - val_accuracy: 0.3333 - val_loss: 1.4776
Epoch 19/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 69ms/step - accuracy: 0.7023 - loss:
0.8964 - val_accuracy: 0.2734 - val_loss: 1.6143
Epoch 20/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5625 - loss:
0.7265 - val_accuracy: 0.3333 - val_loss: 1.7467
```



Plot a graph to show the learning curves (i.e., x-axis: number of epochs; y-axis: training and validation accuracy - 2 curves) (1 points)

```python
import matplotlib.pyplot as plt

# Plotting the learning curves for training and validation accuracy
```

```
plt.figure(figsize=(8, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

Perform ONE of the following experiment below ((a), (b) or (c)) based on the last digit of your Rowan Banner ID (1 point):

(a) Train the CNN using 2 other filter sizes: 5 × 5 and 7 × 7 for the 2nd convolution layer (i) with all other parameters unchanged

(b) Train the CNN using 2 other number of filters: 8 and 16 for the 2nd convolution layer (i) with all other parameters unchanged

(c) Train the CNN using 2 other number of nodes in the hidden layer (iv): 4 and 16 with all other parameters unchanged If the last digit is {0, 1, 2, 3}, do (a). If the last digit is {4, 5, 6}, do (b). If the last digit is {7, 8, 9}, do (c). State your Rowan Banner ID in your submission so that we know which experiment you are doing.

Banner ID is 916496886.

```
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
Flatten, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Path to your dataset
data_dir = 'D:\\Data Mining\\Programming Assignment - 1\\Data Files\\
Images'

train_datagen = ImageDataGenerator(rescale=1./255,
validation_split=0.2)

# Training data generator
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='training')

# Validation data generator
validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='validation')

# Function to create model
def create_model(num_filters_second_layer):
    model = Sequential([
        Input(shape=(150, 150, 3)),
        Conv2D(8, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(num_filters_second_layer, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(8, activation='relu'),
        Dense(4, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Experiment with 8 filters in the second convolution layer
model_8_filters = create_model(8)
history_8_filters = model_8_filters.fit(
    train_generator,
```

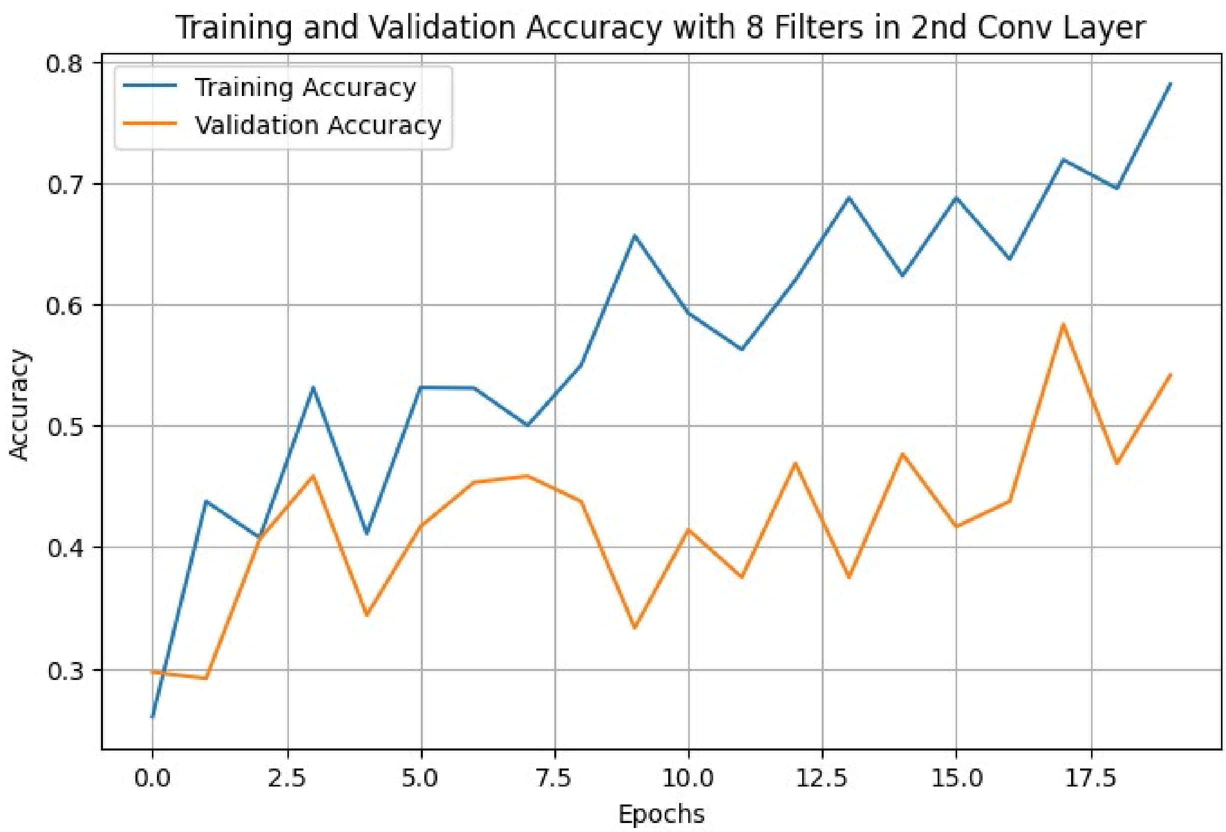
```
    steps_per_epoch=train_generator.samples //
train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples //
validation_generator.batch_size,
    epochs=20)

# Experiment with 16 filters in the second convolution layer
model_16_filters = create_model(16)
history_16_filters = model_16_filters.fit(
    train_generator,
    steps_per_epoch=train_generator.samples //
train_generator.batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples //
validation_generator.batch_size,
    epochs=20)

# Plotting results
def plot_results(history, title):
    plt.figure(figsize=(8, 5))
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()

# Plot results for 8 filters
plot_results(history_8_filters, 'Training and Validation Accuracy with
8 Filters in 2nd Conv Layer')

# Plot results for 16 filters
plot_results(history_16_filters, 'Training and Validation Accuracy
with 16 Filters in 2nd Conv Layer')
```

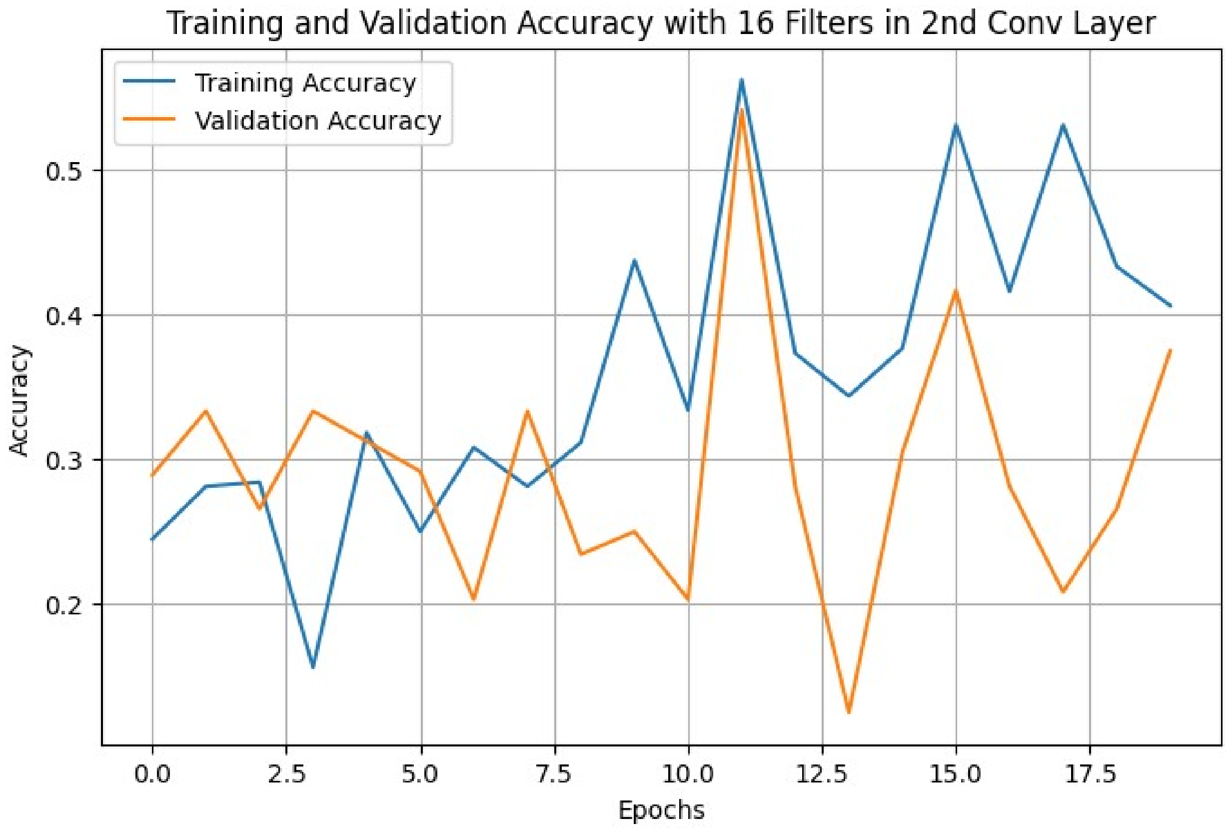
```
Found 616 images belonging to 4 classes.
Found 152 images belonging to 4 classes.
Epoch 1/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 76ms/step - accuracy: 0.2986 - loss:
1.3835 - val_accuracy: 0.2969 - val_loss: 1.3608
Epoch 2/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4375 - loss:
1.3425 - val_accuracy: 0.2917 - val_loss: 1.3307
Epoch 3/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 64ms/step - accuracy: 0.3712 - loss:
1.3234 - val_accuracy: 0.4062 - val_loss: 1.2889
```

```
Epoch 4/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5312 - loss:
1.2202 - val_accuracy: 0.4583 - val_loss: 1.3997
Epoch 5/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.4432 - loss:
1.2338 - val_accuracy: 0.3438 - val_loss: 1.3037
Epoch 6/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5312 - loss:
1.0573 - val_accuracy: 0.4167 - val_loss: 1.3191
Epoch 7/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - accuracy: 0.5075 - loss:
1.1623 - val_accuracy: 0.4531 - val_loss: 1.2386
Epoch 8/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5000 - loss:
1.0350 - val_accuracy: 0.4583 - val_loss: 1.3162
Epoch 9/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 82ms/step - accuracy: 0.4940 - loss:
1.1607 - val_accuracy: 0.4375 - val_loss: 1.2221
Epoch 10/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6562 - loss:
0.8872 - val_accuracy: 0.3333 - val_loss: 1.2783
Epoch 11/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.6244 - loss:
0.9856 - val_accuracy: 0.4141 - val_loss: 1.2880
Epoch 12/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5625 - loss:
1.0694 - val_accuracy: 0.3750 - val_loss: 1.2120
Epoch 13/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 68ms/step - accuracy: 0.5951 - loss:
0.9944 - val_accuracy: 0.4688 - val_loss: 1.2311
Epoch 14/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6875 - loss:
0.8692 - val_accuracy: 0.3750 - val_loss: 1.2667
Epoch 15/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 76ms/step - accuracy: 0.6269 - loss:
0.8891 - val_accuracy: 0.4766 - val_loss: 1.2378
Epoch 16/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6875 - loss:
0.8890 - val_accuracy: 0.4167 - val_loss: 1.4225
Epoch 17/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.6501 - loss:
0.8121 - val_accuracy: 0.4375 - val_loss: 1.3020
Epoch 18/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7188 - loss:
0.7001 - val_accuracy: 0.5833 - val_loss: 1.1485
Epoch 19/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 76ms/step - accuracy: 0.7014 - loss:
0.8188 - val_accuracy: 0.4688 - val_loss: 1.2993
Epoch 20/20
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7812 - loss:
0.7077 - val_accuracy: 0.5417 - val_loss: 1.2160
Epoch 1/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 104ms/step - accuracy: 0.2518 - loss:
1.4076 - val_accuracy: 0.2891 - val_loss: 1.3860
Epoch 2/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.2812 - loss:
1.3857 - val_accuracy: 0.3333 - val_loss: 1.3855
Epoch 3/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 92ms/step - accuracy: 0.2478 - loss:
1.3856 - val_accuracy: 0.2656 - val_loss: 1.3846
Epoch 4/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1562 - loss:
1.3876 - val_accuracy: 0.3333 - val_loss: 1.3851
Epoch 5/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 75ms/step - accuracy: 0.3233 - loss:
1.3710 - val_accuracy: 0.3125 - val_loss: 1.3815
Epoch 6/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2500 - loss:
1.3805 - val_accuracy: 0.2917 - val_loss: 1.3860
Epoch 7/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 99ms/step - accuracy: 0.3336 - loss:
1.3679 - val_accuracy: 0.2031 - val_loss: 1.4062
Epoch 8/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2812 - loss:
1.3251 - val_accuracy: 0.3333 - val_loss: 1.3640
Epoch 9/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step - accuracy: 0.3138 - loss:
1.3238 - val_accuracy: 0.2344 - val_loss: 1.4021
Epoch 10/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4375 - loss:
1.3007 - val_accuracy: 0.2500 - val_loss: 1.3597
Epoch 11/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 72ms/step - accuracy: 0.3125 - loss:
1.2998 - val_accuracy: 0.2031 - val_loss: 1.4027
Epoch 12/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5625 - loss:
1.3139 - val_accuracy: 0.5417 - val_loss: 1.3353
Epoch 13/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 71ms/step - accuracy: 0.3835 - loss:
1.2790 - val_accuracy: 0.2812 - val_loss: 1.3965
Epoch 14/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3438 - loss:
1.3113 - val_accuracy: 0.1250 - val_loss: 1.4343
Epoch 15/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.3936 - loss:
1.2315 - val_accuracy: 0.3047 - val_loss: 1.3730
Epoch 16/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5312 - loss:
```

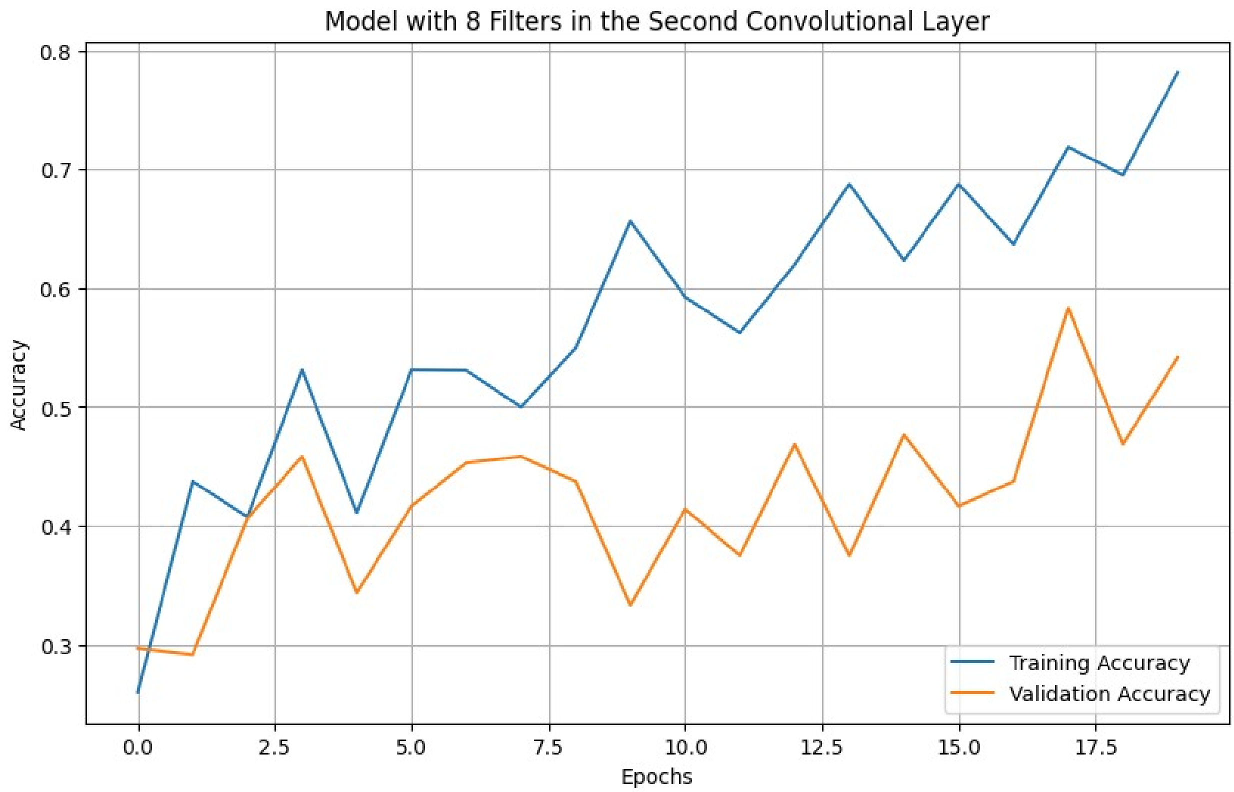
```
1.2583 - val_accuracy: 0.4167 - val_loss: 1.3378
Epoch 17/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 73ms/step - accuracy: 0.4584 - loss:
1.2284 - val_accuracy: 0.2812 - val_loss: 1.4298
Epoch 18/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5312 - loss:
1.2634 - val_accuracy: 0.2083 - val_loss: 1.5520
Epoch 19/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 70ms/step - accuracy: 0.3969 - loss:
1.1926 - val_accuracy: 0.2656 - val_loss: 1.5458
Epoch 20/20
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4062 - loss:
1.1409 - val_accuracy: 0.3750 - val_loss: 1.3150
```

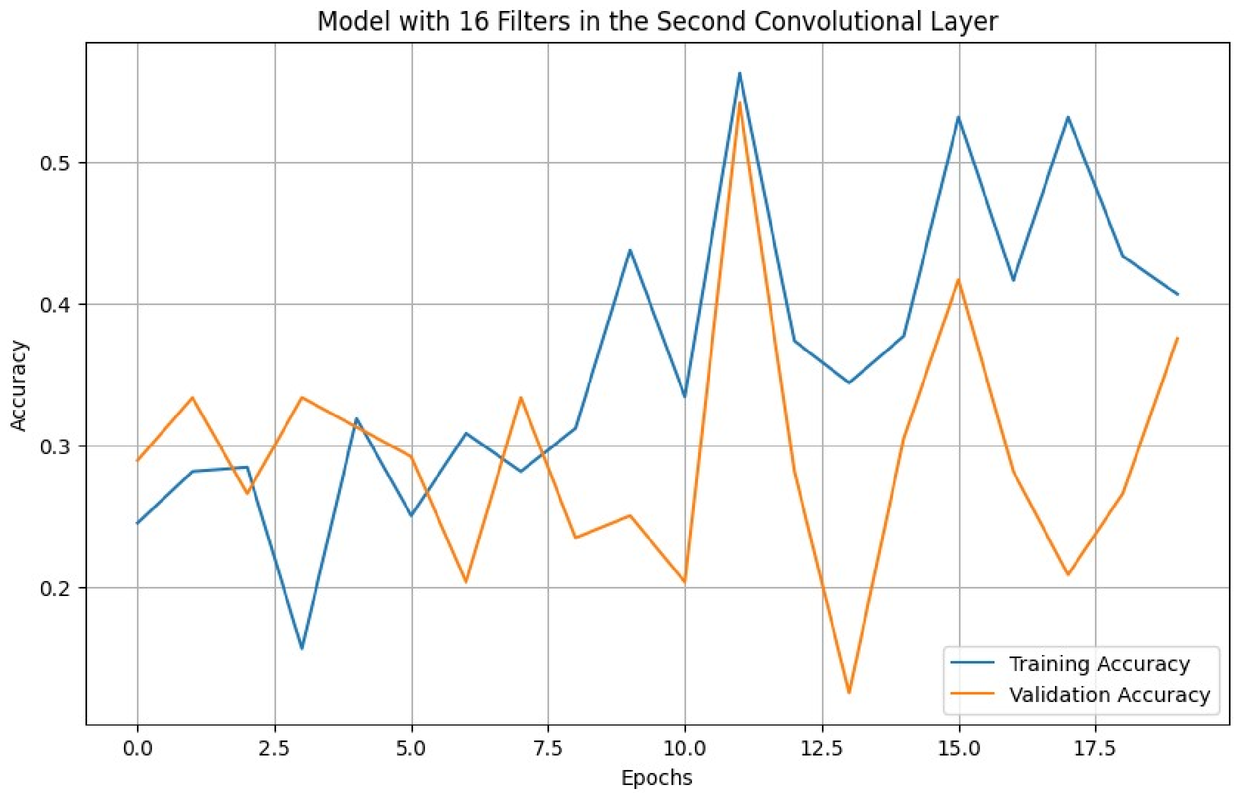





```
Plot the learning curves (i.e., x-axis: number of epochs; y-axis:
training and validation accuracy - 2 curves) for the classification
models using the above 2 different parameter values (1 points)

import matplotlib.pyplot as plt

# Function to plot learning curves
def plot_learning_curves(history, title):
    plt.figure(figsize=(10, 6))
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()

# Plot learning curves for the model with 8 filters in the second
convolution layer
plot_learning_curves(history_8_filters, 'Model with 8 Filters in the
Second Convolutional Layer')
```

```
# Plot learning curves for the model with 16 filters in the second convolution layer
plot_learning_curves(history_16_filters, 'Model with 16 Filters in the Second Convolutional Layer')
```





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. (1 point)

With reference to the general behavior of the convolutional neural networks when number of filters are changed, he following could be inferred on the relative performances of the first model assuming that the second convolutional layer initially had lesser filters and the two constructed with 8 and 16 filters respectively in the second convolutional layer

Performance Analysis

1. Model with 4 Filters (Original Model): This power has the least number of filters, and therefore may not have sufficient capability to detect intricate structures within the data which makes the specific task more of a possibility to underfitting if the given work is rather complicated. If the accuracy of this model both for training data and the validation set is lower compared to the other two models or if the rate of raise of the accuracies is slow this will indicate that the model is inadequate to the complexity of the data.

2. Model with 8 Filters: The use of more filters in the convolutional layer enables a model to learn many details, and possibly increases the model accuracy without significant complexity that can enhance overfitting. If it would show that this model has a better training-validation accuracy split, and the accuracies are both greater than the first model, but not significantly so, then this model is perfect – this model has good capacity, but it will not overfit too much.

3. Model with 16 Filters: This model has more workload than the other models in the analysis of medical data of patients. It can give very detailed features and probably the greatest training accuracy possible, but it can easily overfit and if the validation accuracy begins to become significantly less than the training accuracy. An overly large difference between the sizes of validation accuracy and of training accuracy—the learning curve of the model—would mean that the model has overfitted.

Observational Insights

- Learning Dynamics: An important implication for models with higher filter counts should be faster learning as well as achieving better accuracies in the initial stages. How fast the accuracy in each of the models increases can tell us something about efficiency and efficacy of the models.
- Stability and Convergence: For both the training and the validation set, an upward sloping and non-flattening curve is desirable, but one that is at a high level of accuracy. Shifts or disparities in between these curves can provide signs of learning dynamics problem.
- Optimal Configuration: Of course, this may also depend on the type of particular data set and the complexity of the problem to be solved. Depending on the actual structure of datasets, it is likely that fewer filters would be enough for simple datasets whereas more complicated structures of the datasets may require elaborate filter mechanisms.

3)Text Classification by fine-tuning LLM model

Plot the two learning curves - training and validation (i.e., x-axis: number of epochs; y-axis: losses) for 5 epochs. (1 point) Using the approach to compute accuracy (i.e., all labels must match) in the tutorial, what is the test accuracy? (0.5 points) Modify the accuracy such that a prediction is correct as long as one label matches. What is the test accuracy? (0.5 points)

```
import torch
from torch.utils.data import Dataset, DataLoader
from torch.optim import AdamW
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import get_linear_schedule_with_warmup
import json
import numpy as np
import matplotlib.pyplot as plt
import random

# Setup GPU/CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load datasets with sampling
#I'm using only 10% of the dataset because my Laptop is unable to
handle the larger amount data and trainig the model
#I hope you understand the scenario
# 0.10 refers to 10% of data is used
def load_data(filepath, sample_fraction=0.10):
    with open(filepath, 'r') as file:
        lines = file.readlines()
        sampled_lines = random.sample(lines, int(len(lines) *
sample_fraction))
        data = [json.loads(line) for line in sampled_lines]
    return data
'''
# Load datasets without sampling
def load_data(filepath):
    with open(filepath, 'r') as file:
        data = [json.loads(line) for line in file]
    return data
'''

train_data = load_data(r'D:\Data Mining\Programming Assignment - 3\
Data Files\student_5\train.json')
val_data = load_data(r'D:\Data Mining\Programming Assignment - 3\Data
Files\student_5\validation.json')
test_data = load_data(r'D:\Data Mining\Programming Assignment - 3\Data
Files\student_5\test.json')

# Define dataset class
class TweetDataset(Dataset):
```

```
    def __init__(self, data, tokenizer, max_len):
        self.tokenizer = tokenizer
        self.data = data
        self.max_len = max_len

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        item = self.data[idx]
        tweet = item['Tweet']
        labels = [int(item[key]) for key in ['anger', 'anticipation',
'disgust', 'fear', 'joy', 'love',
                                             'optimism', 'pessimism',
'sadness', 'surprise', 'trust']]
        encoding = self.tokenizer.encode_plus(
            tweet,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='pt',
            truncation=True
        )
        return {
            'tweet_text': tweet,
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(labels, dtype=torch.float)
        }

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

# Dataset and DataLoader
max_len = 256
batch_size = 16

train_dataset = TweetDataset(train_data, tokenizer, max_len)
val_dataset = TweetDataset(val_data, tokenizer, max_len)
test_dataset = TweetDataset(test_data, tokenizer, max_len)

train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

# Load BERT model
model = BertForSequenceClassification.from_pretrained('bert-base-
```

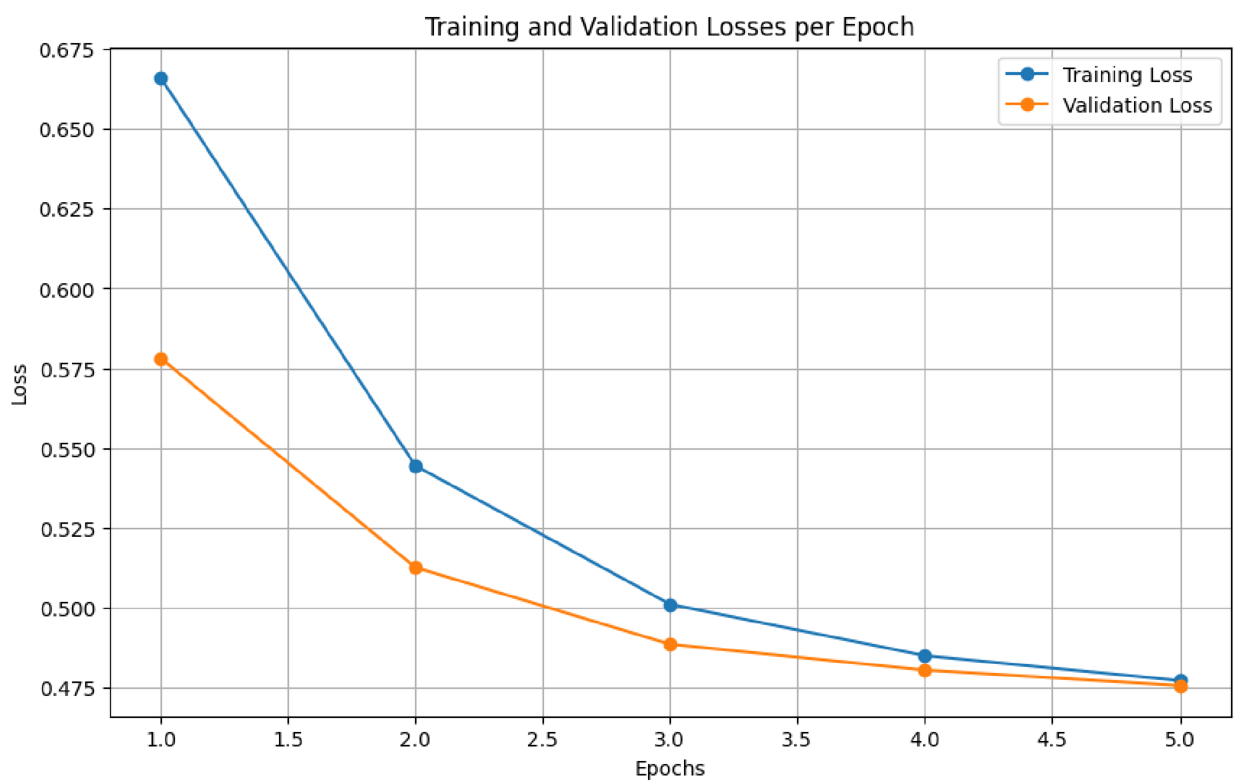
```
uncased', num_labels=11).to(device)

# Optimizer and scheduler
optimizer = AdamW(model.parameters(), lr=2e-5)
total_steps = len(train_loader) * 5  # number of epochs
scheduler = get_linear_schedule_with_warmup(optimizer,
num_warmup_steps=0, num_training_steps=total_steps)

# Lists to store losses for plotting
training_losses = []
validation_losses = []

# Training loop
def train_epoch(model, data_loader, optimizer, device, scheduler):
    model.train()
    losses = []
    for d in data_loader:
        input_ids = d["input_ids"].to(device)
        attention_mask = d["attention_mask"].to(device)
        labels = d["labels"].to(device)

        outputs = model(input_ids=input_ids,
attention_mask=attention_mask, labels=labels)
        loss = outputs.loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        scheduler.step()

        losses.append(loss.item())
    return np.mean(losses)

# Evaluation function
def evaluate(model, data_loader, device):
    model.eval()
    all_preds = []
    all_labels = []
    losses = []
    with torch.no_grad():
        for d in data_loader:
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            labels = d["labels"].to(device)

            outputs = model(input_ids=input_ids,
attention_mask=attention_mask, labels=labels)
            logits = outputs.logits
            loss = outputs.loss
            losses.append(loss.item())
```

```
            preds = torch.sigmoid(logits).cpu().numpy() > 0.5
            all_preds.append(preds)
            all_labels.append(labels.cpu().numpy() > 0.5)  # Ensure
this is also binary

    all_preds = np.vstack(all_preds).astype(int)  # Ensure integer
type for bitwise operations
    all_labels = np.vstack(all_labels).astype(int)

    # Compute metrics
    exact_match_accuracy = np.mean((all_preds ==
all_labels).all(axis=1))
    at_least_one_match_accuracy =
np.mean(np.any(np.bitwise_and(all_preds, all_labels), axis=1))  #
Using np.bitwise_and

    return np.mean(losses), exact_match_accuracy,
at_least_one_match_accuracy

# Fine-tuning the model
for epoch in range(5):
    train_loss = train_epoch(model, train_loader, optimizer, device,
scheduler)
    training_losses.append(train_loss)
    print(f"Epoch {epoch + 1}, Train Loss: {train_loss:.4f}")

    val_loss, _, _ = evaluate(model, val_loader, device)
    validation_losses.append(val_loss)
    print(f"Epoch {epoch + 1}, Validation Loss: {val_loss:.4f}")

# Plot learning curves
plt.figure(figsize=(10, 6))
plt.plot(epochs, training_losses, label='Training Loss', marker='o')
plt.plot(epochs, validation_losses, label='Validation Loss',
marker='o')
plt.title('Training and Validation Losses per Epoch')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Evaluate on the test set
test_loss, exact_match_accuracy, at_least_one_match_accuracy =
evaluate(model, test_loader, device)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy (Exact Match): {exact_match_accuracy:.4f}")
print(f"Test Accuracy (At Least One Match):
{at_least_one_match_accuracy:.4f}")
```

```python
# Save the model and tokenizer
model.save_pretrained('fine_tuned_bert_model')
tokenizer.save_pretrained('fine_tuned_bert_model')
```

```
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, Train Loss: 0.6661
Epoch 1, Validation Loss: 0.5782
Epoch 2, Train Loss: 0.5445
Epoch 2, Validation Loss: 0.5126
Epoch 3, Train Loss: 0.5011
Epoch 3, Validation Loss: 0.4886
Epoch 4, Train Loss: 0.4850
Epoch 4, Validation Loss: 0.4805
Epoch 5, Train Loss: 0.4772
Epoch 5, Validation Loss: 0.4757
```




```
Test Loss: 0.4873
Test Accuracy (Exact Match): 0.0067
Test Accuracy (At Least One Match): 0.0000
```

```
('fine_tuned_bert_model\\tokenizer_config.json',
 'fine_tuned_bert_model\\special_tokens_map.json',
 'fine_tuned_bert_model\\vocab.txt',
 'fine_tuned_bert_model\\added_tokens.json')
```