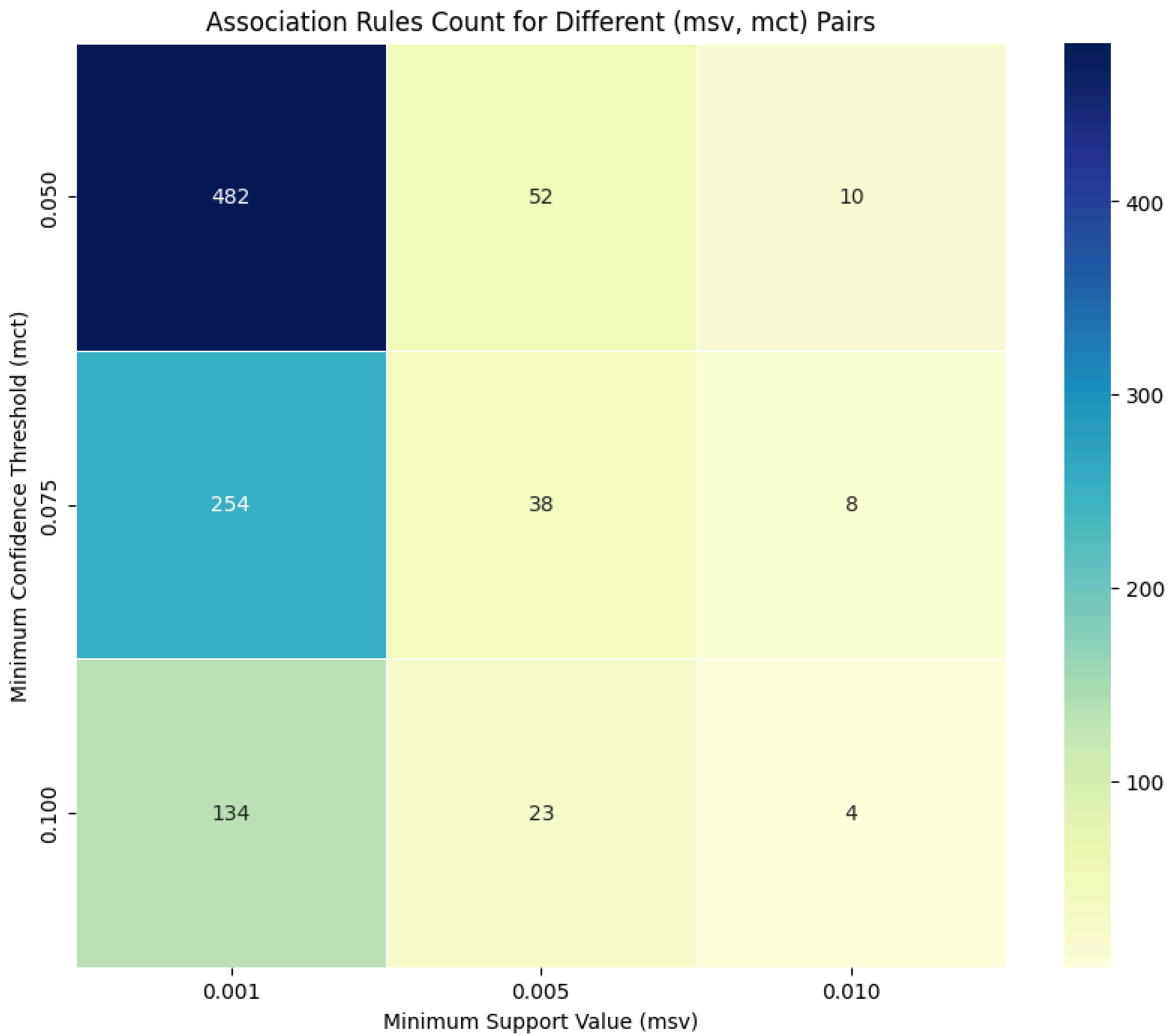
In [33]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
from collections import Counter

data = pd.read_csv("C:/Users/Gowtham reddy/Downloads/Grocery_Items_29.csv", header=None)

data = data.fillna('').astype(str)

transactions = [[item for item in row if item != ''] for row in data.values.tolist()]

unique_items = set(item for transaction in transactions for item in transaction)
print(f"Number of unique items: {len(unique_items)}")
print(f"Number of records: {len(transactions)}")

item_counts = Counter(item for transaction in transactions for item in transaction)
most_popular_item = max(item_counts, key=item_counts.get)
print(f"Most popular item: {most_popular_item}")
print(f"Number of transactions containing the most popular item: {item_counts[most_popular_item]}")

te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets_d = apriori(df, min_support=0.01, use_colnames=True)
rules_d = association_rules(frequent_itemsets_d, metric="confidence", min_threshold=0.08, num_itemsets=2)
print("\nAssociation rules with min_support=0.01 and min_confidence=0.08:")
print(rules_d)

msv_values = [0.001, 0.005, 0.01]
mct_values = [0.05, 0.075, 0.1]
heatmap_data = []
total_rules = 0

for mct in mct_values:
    row = []
    for msv in msv_values:
        frequent_itemsets = apriori(df, min_support=msv, use_colnames=True)
```

```
        rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=mct, num_itemsets=2)
        num_rules = len(rules)
        row.append(num_rules)
        total_rules += num_rules
    heatmap_data.append(row)

heatmap_df = pd.DataFrame(heatmap_data,
                          columns=[f'{msv:.3f}' for msv in msv_values],
                          index=[f'{mct:.3f}' for mct in mct_values])

plt.figure(figsize=(10,8))
sns.heatmap(heatmap_df, annot=True, cmap="YlGnBu", fmt="d", linewidths=0.5, cbar=True)
plt.title("Association Rules Count for Different (msv, mct) Pairs")
plt.xlabel("Minimum Support Value (msv)")
plt.ylabel("Minimum Confidence Threshold (mct)")
plt.show()

print(f"\nTotal number of association rules extracted from the dataset: {total_rules}")
```

```
Number of unique items: 177
Number of records: 8001
Most popular item: whole milk
Number of transactions containing the most popular item: 1360

Association rules with min_support=0.01 and min_confidence=0.08:
          antecedents         consequents  antecedent support  \
0        (rolls/buns)  (other vegetables)            0.112486
1  (other vegetables)        (rolls/buns)            0.123610
2        (whole milk)  (other vegetables)            0.161480
3  (other vegetables)        (whole milk)            0.123610
4        (whole milk)        (rolls/buns)            0.161480
5        (rolls/buns)        (whole milk)            0.112486
6              (soda)        (whole milk)            0.093988
7            (yogurt)        (whole milk)            0.084239

   consequent support   support  confidence      lift  representativity  \
0            0.123610  0.010124    0.090000  0.728099               1.0
1            0.112486  0.010124    0.081901  0.728099               1.0
2            0.123610  0.014748    0.091331  0.738869               1.0
3            0.161480  0.014748    0.119312  0.738869               1.0
4            0.112486  0.014998    0.092879  0.825697               1.0
5            0.161480  0.014998    0.133333  0.825697               1.0
6            0.161480  0.011749    0.125000  0.774091               1.0
7            0.161480  0.011124    0.132047  0.817734               1.0

   leverage  conviction  zhangs_metric   jaccard  certainty  kulczynski
0 -0.003781    0.963066      -0.296156  0.044801  -0.038350    0.085950
1 -0.003781    0.966687      -0.298792  0.044801  -0.034461    0.085950
2 -0.005212    0.964477      -0.296508  0.054554  -0.036831    0.105322
3 -0.005212    0.952120      -0.287378  0.054554  -0.050288    0.105322
4 -0.003166    0.978386      -0.201119  0.057915  -0.022092    0.113106
5 -0.003166    0.967523      -0.192150  0.057915  -0.033567    0.113106
6 -0.003429    0.958309      -0.243635  0.048205  -0.043505    0.098878
7 -0.002479    0.966090      -0.195751  0.047416  -0.035100    0.100466
```


Association Rules Count for Different (msv, mct) Pairs
Minimum Confidence Threshold (mct)
0.050
0.075
0.100
482
52
10
254
38
8
134
23
4
0.001
0.005
0.010
Minimum Support Value (msv)
400
300
200
100

```
Total number of association rules extracted from the dataset: 1005
```

In [34]:
```python
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
import os
```

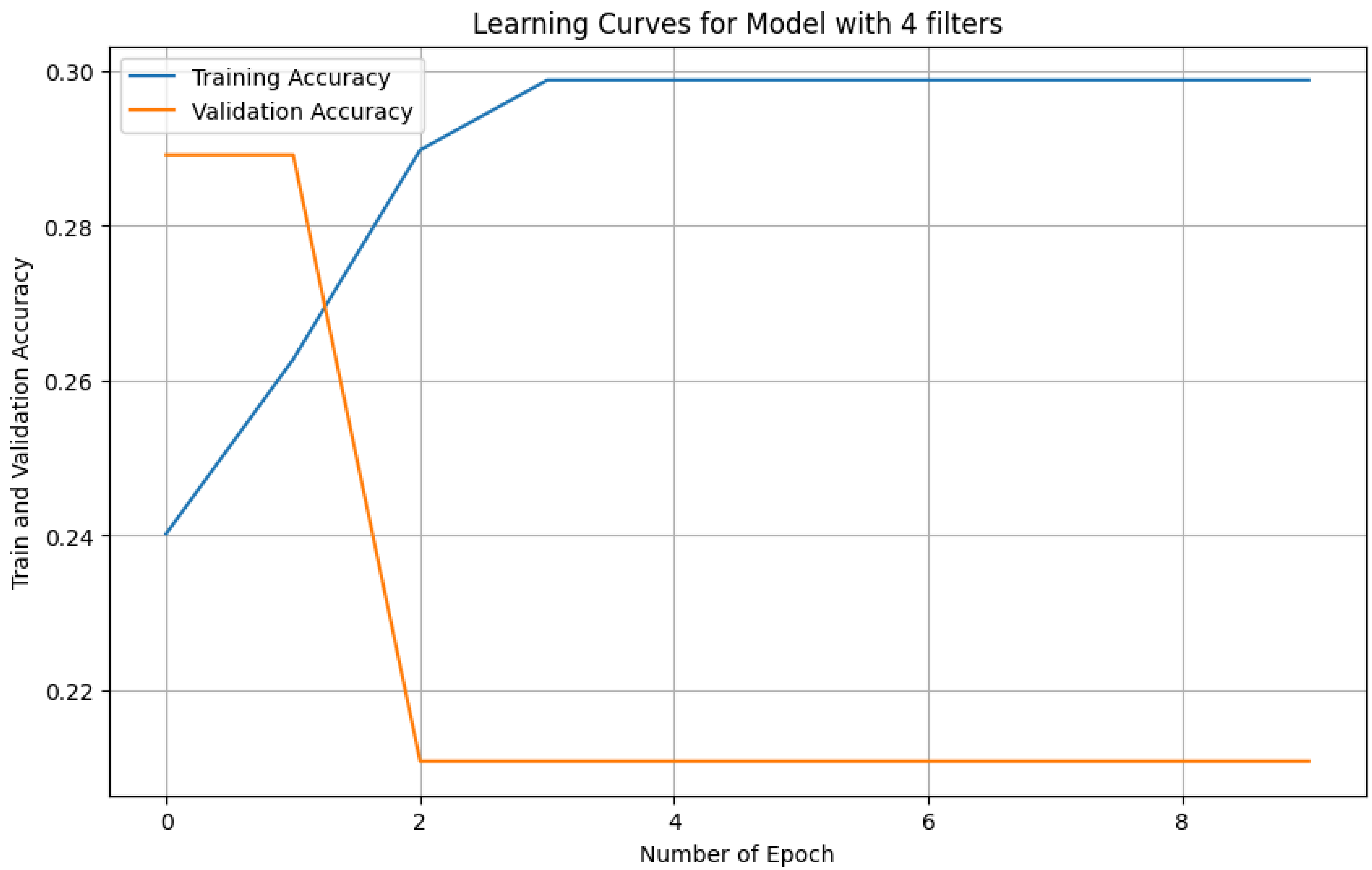
BANNER ID: 916472365

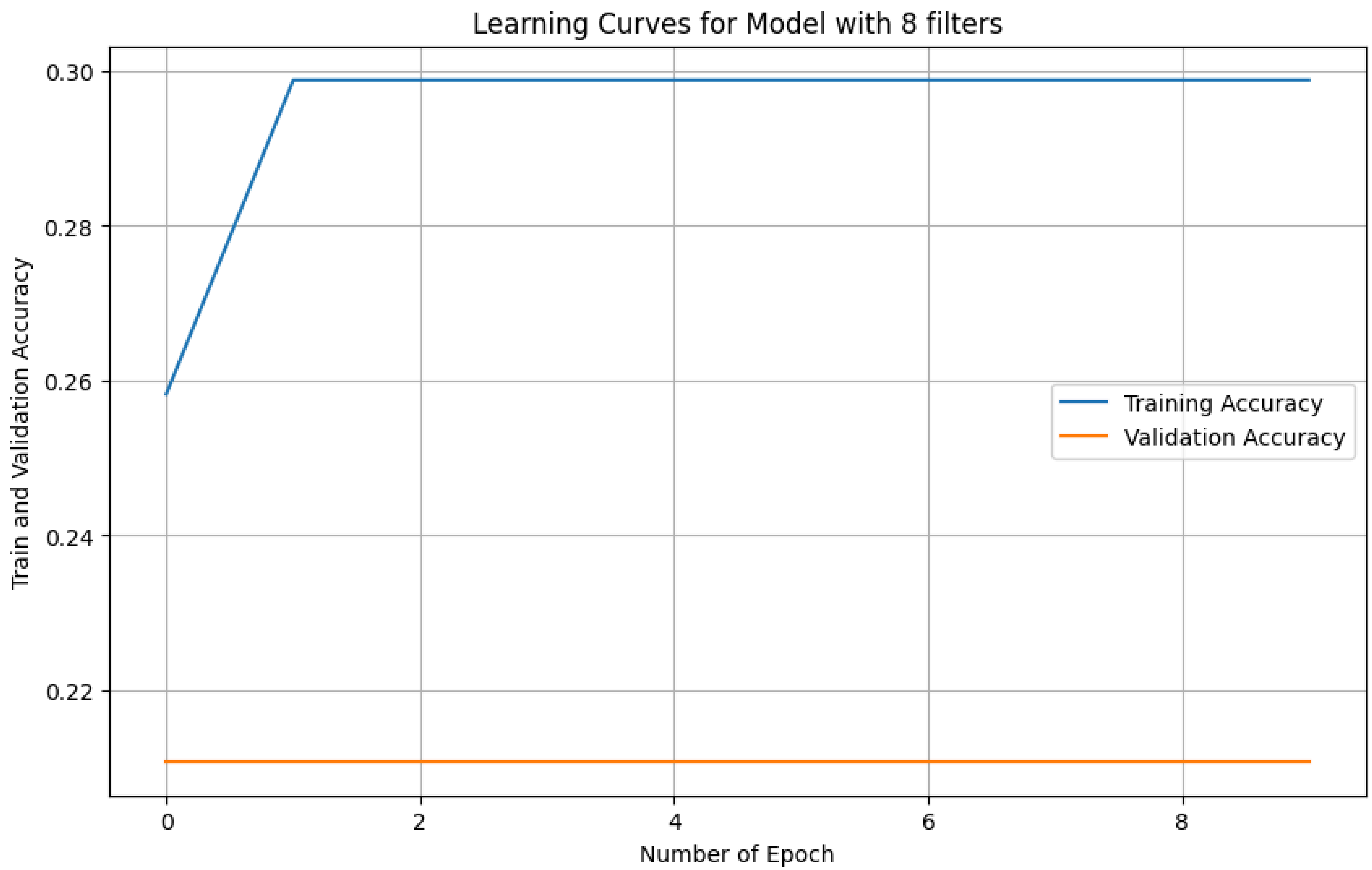
In [47]:
```python
data_dir = 'C:/Users/Gowtham reddy/DM1/Cropped'

# Image parameters
img_height = 180
img_width = 180
batch_size = 32

# Load and preprocess the dataset
train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

# Function to create and train the model
def create_and_train_model(num_filters_second_conv):
    model = models.Sequential([
        layers.Conv2D(8, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(num_filters_second_conv, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
```

```
        layers.Flatten(),
        layers.Dense(8, activation='relu'),
        layers.Dense(4, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    history = model.fit(
        train_ds,
        validation_data=val_ds,
        epochs=10
    )
    return history

# Train models with different numbers of filters in the second conv layer
filter_sizes = [4, 8, 16]
histories = []

for filters in filter_sizes:
    print(f"\nTraining model with {filters} filters in the second convolutional layer")
    history = create_and_train_model(filters)
    histories.append(history)

# Plotting learning curves individually
for i, history in enumerate(histories):
    plt.figure(figsize=(10, 6))
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'Learning Curves for Model with {filter_sizes[i]} filters')
    plt.xlabel('Number of Epoch')
    plt.ylabel('Train and Validation Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()

# Compare final validation accuracies
best_model_index = np.argmax([h.history['val_accuracy'][-1] for h in histories])
best_filters = filter_sizes[best_model_index]
print(f"\nThe model with {best_filters} filters in the second convolutional layer performed best.")
```

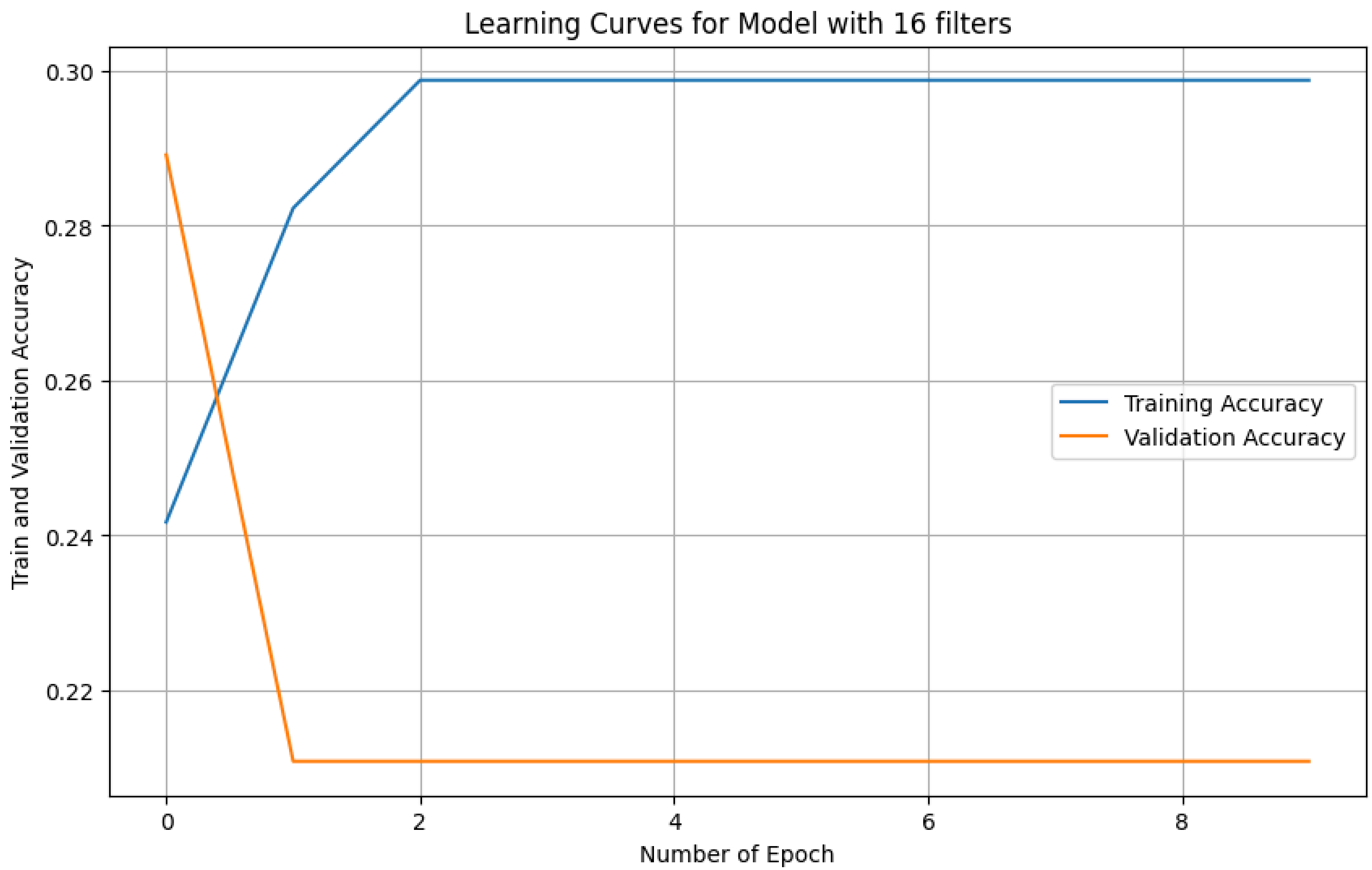
```
Found 832 files belonging to 4 classes.
Using 666 files for training.
Found 832 files belonging to 4 classes.
Using 166 files for validation.

Training model with 4 filters in the second convolutional layer
Epoch 1/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.2641 - loss: 16.8194 - val_accuracy: 0.2892 - val_loss: 1.3863
Epoch 2/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.2731 - loss: 1.3861 - val_accuracy: 0.2892 - val_loss: 1.3863
Epoch 3/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.2811 - loss: 1.3852 - val_accuracy: 0.2108 - val_loss: 1.3862
Epoch 4/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.2980 - loss: 1.3835 - val_accuracy: 0.2108 - val_loss: 1.3863
Epoch 5/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.2982 - loss: 1.3819 - val_accuracy: 0.2108 - val_loss: 1.3865
Epoch 6/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.2877 - loss: 1.3821 - val_accuracy: 0.2108 - val_loss: 1.3866
Epoch 7/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.2857 - loss: 1.3806 - val_accuracy: 0.2108 - val_loss: 1.3868
Epoch 8/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.2921 - loss: 1.3792 - val_accuracy: 0.2108 - val_loss: 1.3870
Epoch 9/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.2959 - loss: 1.3784 - val_accuracy: 0.2108 - val_loss: 1.3873
Epoch 10/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.3229 - loss: 1.3768 - val_accuracy: 0.2108 - val_loss: 1.3877

Training model with 8 filters in the second convolutional layer
Epoch 1/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.2513 - loss: 25.8168 - val_accuracy: 0.2108 - val_loss: 1.3865
Epoch 2/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.3057 - loss: 1.3853 - val_accuracy: 0.2108 - val_loss: 1.3866
Epoch 3/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.2791 - loss: 1.3847 - val_accuracy: 0.2108 - val_loss: 1.3867
Epoch 4/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 0.2847 - loss: 1.3841 - val_accuracy: 0.2108 - val_loss: 1.3869
Epoch 5/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.2946 - loss: 1.3827 - val_accuracy: 0.2108 - val_loss: 1.3871
Epoch 6/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 50ms/step - accuracy: 0.2770 - loss: 1.3828 - val_accuracy: 0.2108 - val_loss: 1.3872
Epoch 7/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.2880 - loss: 1.3806 - val_accuracy: 0.2108 - val_loss: 1.3876
```

```
Epoch 8/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.2782 - loss: 1.3823 - val_accuracy: 0.2108 - val_loss: 1.3878
Epoch 9/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.2982 - loss: 1.3781 - val_accuracy: 0.2108 - val_loss: 1.3880
Epoch 10/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.3112 - loss: 1.3773 - val_accuracy: 0.2108 - val_loss: 1.3885

Training model with 16 filters in the second convolutional layer
Epoch 1/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.2601 - loss: 268.3971 - val_accuracy: 0.2892 - val_loss: 1.3864
Epoch 2/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 59ms/step - accuracy: 0.2708 - loss: 1.3862 - val_accuracy: 0.2108 - val_loss: 1.3864
Epoch 3/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 0.2939 - loss: 1.3854 - val_accuracy: 0.2108 - val_loss: 1.3864
Epoch 4/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 57ms/step - accuracy: 0.2798 - loss: 1.3844 - val_accuracy: 0.2108 - val_loss: 1.3863
Epoch 5/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 0.2918 - loss: 1.3833 - val_accuracy: 0.2108 - val_loss: 1.3864
Epoch 6/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.3067 - loss: 1.3820 - val_accuracy: 0.2108 - val_loss: 1.3864
Epoch 7/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 0.2965 - loss: 1.3820 - val_accuracy: 0.2108 - val_loss: 1.3866
Epoch 8/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 0.2525 - loss: 1.3835 - val_accuracy: 0.2108 - val_loss: 1.3866
Epoch 9/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 0.2814 - loss: 1.3805 - val_accuracy: 0.2108 - val_loss: 1.3868
Epoch 10/10
21/21 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - accuracy: 0.3015 - loss: 1.3795 - val_accuracy: 0.2108 - val_loss: 1.3871
```


Learning Curves for Model with 4 filters
Training Accuracy
Validation Accuracy
0.30
0.28
0.26
0.24
0.22
Train and Validation Accuracy
0
2
4
6
8
Number of Epoch



Learning Curves for Model with 8 filters
0.30
0.28
0.26
0.24
0.22
Train and Validation Accuracy
0
2
4
6
8
Number of Epoch
Training Accuracy
Validation Accuracy




```
The model with 4 filters in the second convolutional layer performed best.
```

First Model: The first model with 4 filters in the second convolutional layer starts with 26.41% accuracy and reaches 32.29% after 10 epochs. However, the validation accuracy remains around 21%, suggesting the model is underfitting, as it struggles to generalize despite slight improvement in training accuracy.

Second Model: The model with 8 filters shows a similar trend, with accuracy increasing from 25.13% to 31.12% over 10 epochs, while the validation accuracy stays at 21%. This indicates underfitting as the increase in filters does not improve generalizatio.

Third Model: The model with 16 filters also demonstrates minimal improvement, with accuracy rising from 26.01% to 30.15%, while validation accuracy remains stuck at 21%. Again, this suggests underfitting, as additional filters do not enhance performance significanuracy.

In [7]:

```
import json
import torch
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from transformers import (
    BertTokenizer,
    BertForSequenceClassification,
    TrainingArguments,
    Trainer
)
from torch.utils.data import Dataset
```

In [9]:

```
import json

def load_dataset(file_path):
    data = []
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            for line in file:
                data.append(json.loads(line.strip()))
        return data
    except Exception as e:
        print(f"Error loading {file_path}: {e}")
        return None

# Load datasets
train_data = load_dataset(r"C:/Users/Gowtham reddy/DM1/train.json")
test_data = load_dataset(r"C:/Users/Gowtham reddy/DM1/test.json")
validation_data = load_dataset(r"C:/Users/Gowtham reddy/DM1/validation.json")
```

In [12]:

```
class DataPreprocessor:
    def __init__(self):
        self.tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
        self.labels = ['anger', 'anticipation', 'disgust', 'fear', 'joy', 'love',
                      'optimism', 'pessimism', 'sadness', 'surprise', 'trust']
```

```
    def preprocess(self, data):
        texts = [item['Tweet'] for item in data]
        labels = [[int(item[label]) for label in self.labels] for item in data]

        encodings = self.tokenizer(
            texts,
            truncation=True,
            padding=True,
            max_length=128,
            return_tensors='pt'
        )

        return encodings, labels

preprocessor = DataPreprocessor()
train_encodings, train_labels = preprocessor.preprocess(train_data)
test_encodings, test_labels = preprocessor.preprocess(test_data)
val_encodings, val_labels = preprocessor.preprocess(validation_data)
```

In [13]:
```
class TweetDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: self.encodings[key][idx] for key in self.encodings}
        item['labels'] = torch.tensor(self.labels[idx], dtype=torch.float)
        return item

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

train_dataset = TweetDataset(train_encodings, train_labels)
val_dataset = TweetDataset(val_encodings, val_labels)
test_dataset = TweetDataset(test_encodings, test_labels)
```

In [15]:
```
model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=11,
    problem_type="multi_label_classification"
)
```

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

In [16]:

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy="epoch"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)
```

```
C:\Users\Gowtham reddy\AppData\Roaming\Python\Python311\site-packages\transformers\training_args.py:1568: FutureWarning: `evaluation_strategy` is deprecated a
nd will be removed in version 4.46 of 🤗 Transformers. Use `eval_strategy` instead
  warnings.warn(
```

In [17]:

```
trainer.train()
```

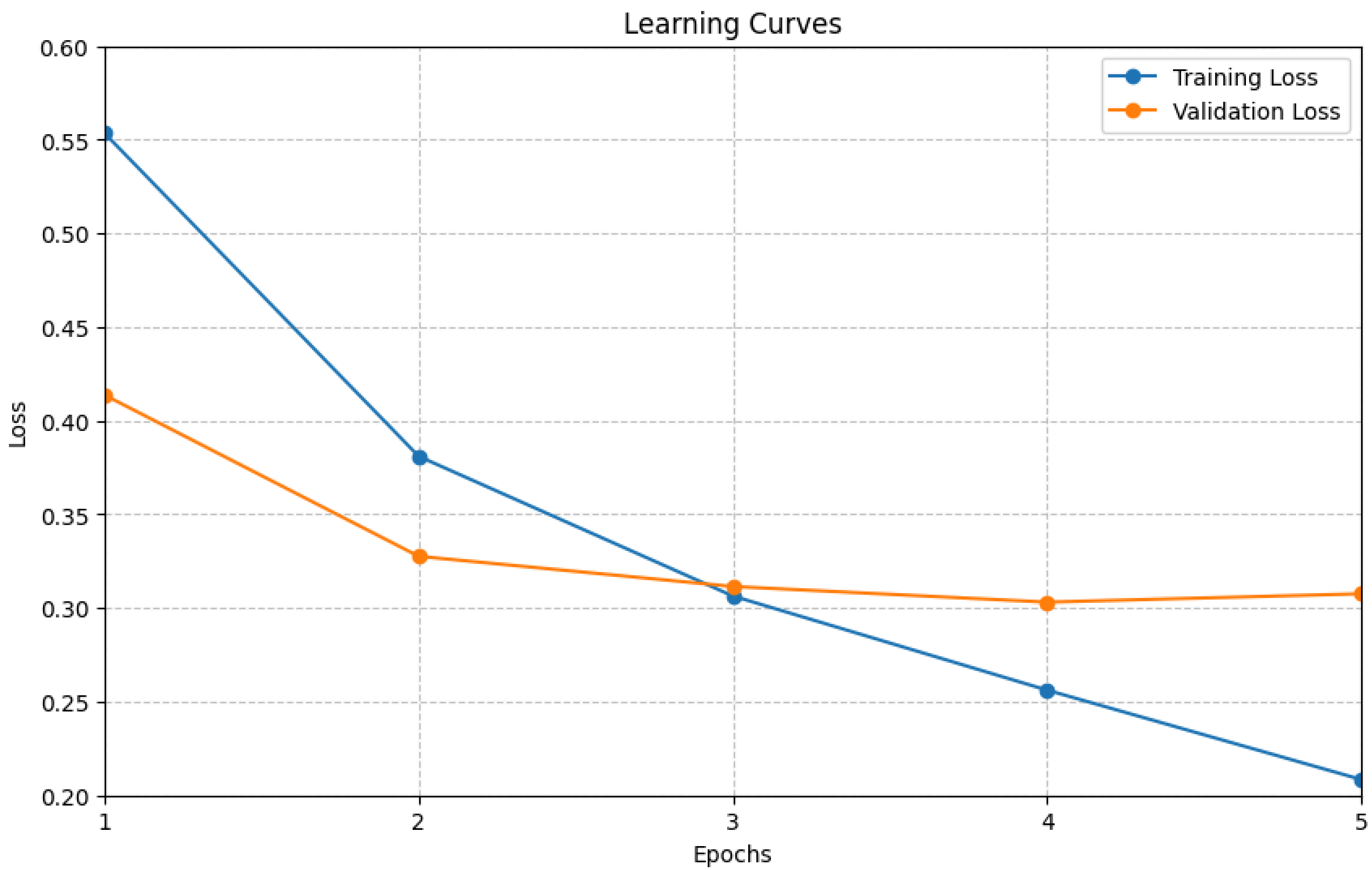
[940/940 29:47, Epoch 5/5]

Epoch	Training Loss	Validation Loss
1	0.425300	0.413736
2	0.341300	0.327625
3	0.279600	0.311525
4	0.217700	0.303173
5	0.204300	0.307617

Out[17]: TrainOutput(global_step=940, training_loss=0.33491256680894405, metrics={'train_runtime': 1789.9422, 'train_samples_per_second': 8.38, 'train_steps_per_second': 0.525, 'total_flos': 547335775890000.0, 'train_loss': 0.33491256680894405, 'epoch': 5.0})

In [25]:

```python
import numpy as np
import matplotlib.pyplot as plt

def plot_learning_curves(trainer):
    # Extract training and validation losses from the trainer's log history
    train_losses = [log['loss'] for log in trainer.state.log_history if 'loss' in log]
    eval_losses = [log['eval_loss'] for log in trainer.state.log_history if 'eval_loss' in log]

    # Ensure we only take losses for the epochs that were completed
    num_epochs = min(len(train_losses), len(eval_losses))

    # If necessary, average training losses over steps per epoch
    steps_per_epoch = len(train_losses) // num_epochs
    train_losses_avg = [np.mean(train_losses[i * steps_per_epoch:(i + 1) * steps_per_epoch]) for i in range(num_epochs)]

    # Ensure train_losses_avg has the same length as eval_losses
    train_losses_avg = train_losses_avg[:num_epochs]

    # Create the plot
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, num_epochs + 1), train_losses_avg, label='Training Loss', marker='o')
    plt.plot(range(1, num_epochs + 1), eval_losses[:num_epochs], label='Validation Loss', marker='o')

    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Learning Curves')
    plt.legend()

    plt.xticks(range(1, num_epochs + 1))  # X-axis ticks from 1 to num_epochs
    plt.xlim(1, num_epochs)  # X-axis limits from 1 to num_epochs
    plt.ylim(0.2, 0.6)  # Set y-axis limits from 0.2 to 0.45

    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

# Call the function to plot
plot_learning_curves(trainer)
```



In [35]:
```
def compute_accuracies(true_labels, pred_labels):
    # Compute accuracy where all labels must match
    strict_accuracy = accuracy_score(true_labels, pred_labels)

    # Compute modified accuracy where at least one label must match
    def one_label_match_accuracy(y_true, y_pred):
        correct = sum(np.any(y_true[i] & y_pred[i]) for i in range(len(y_true)))
        return correct / len(y_true)
```

```python
    modified_accuracy = one_label_match_accuracy(true_labels, pred_labels)

    return strict_accuracy, modified_accuracy

# Assuming you have your test predictions
predictions = trainer.predict(test_dataset)
pred_labels = (predictions.predictions > 0.5).astype(int)

# Get true labels from your test dataset
true_labels = np.array(test_labels)

# Compute both accuracies
strict_acc, modified_acc = compute_accuracies(true_labels, pred_labels)

print(f"Test Accuracy (all labels must match): {strict_acc:.4f}")
print(f"Modified Test Accuracy (at least one label matches): {modified_acc:.4f}")
```

```
Test Accuracy (all labels must match): 0.2780
Modified Test Accuracy (at least one label matches): 0.8173
```