In [ ]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import os
from PIL import Image
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
import seaborn as sns
import tensorflow.keras
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

```
WARNING:tensorflow:From c:\Users\kaasa\AppData\Local\Programs\Python\Python310\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_cr
oss_entropy instead.
```

Question 1

In [ ]:
```
df = pd.read_csv("../Assignment3/Grocery_Items_24.csv")
```

In [ ]:
```
# referred from https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/

transactions = []
for i in range(len(df)):
    transactions.append([str(item) for item in df.iloc[i, :]])

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


min_support = 0.01
frequent_itemsets = apriori(df_encoded, min_support=min_support, use_colnames=True)
```

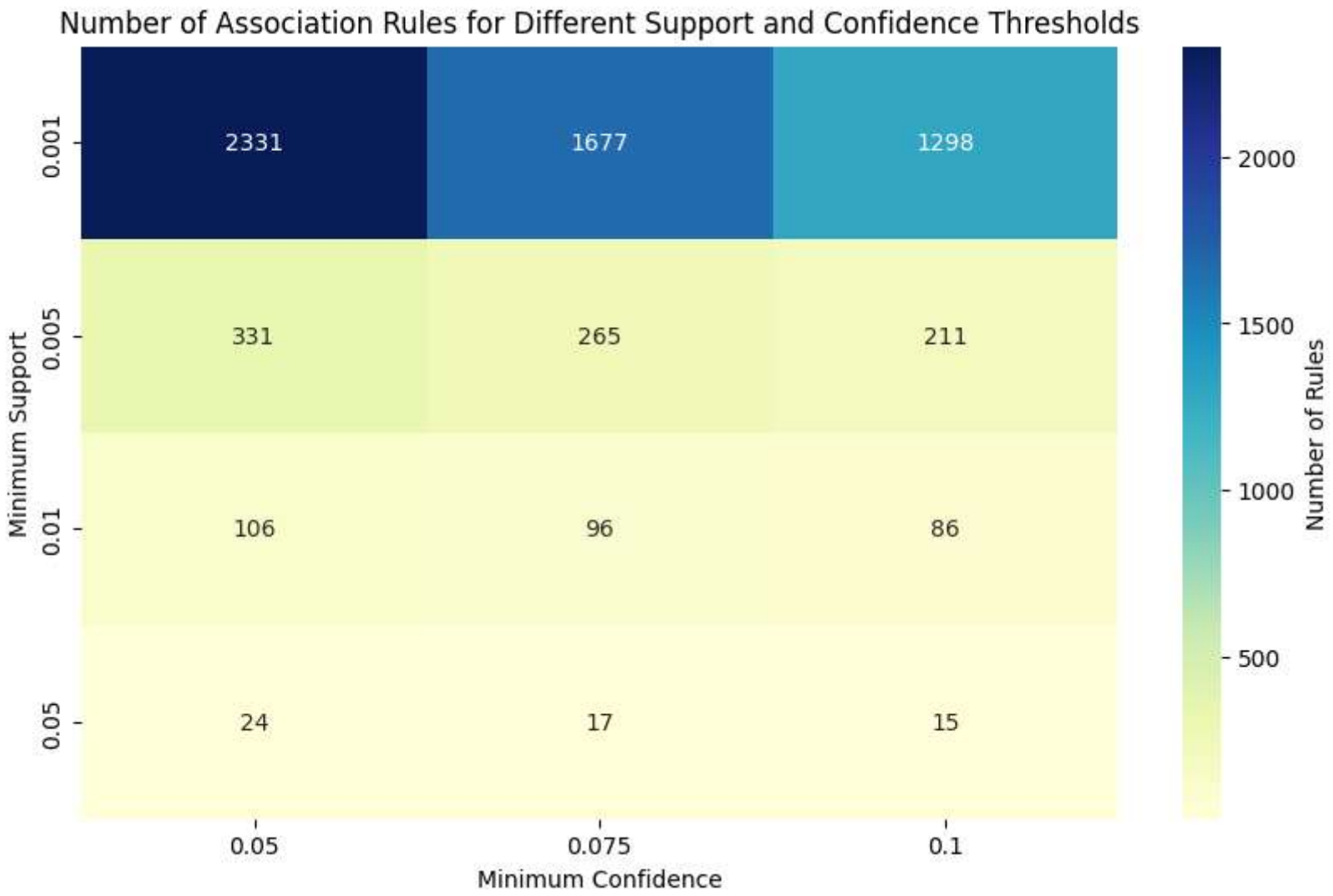
```
# Generating association rules
min_confidence = 0.1
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)

# Print the association rules
print("Association Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence']])
```

```
Association Rules:
           antecedents        consequents   support  confidence
0           (UHT-milk)              (nan)  0.020875    1.000000
1               (beef)              (nan)  0.034250    1.000000
2            (berries)              (nan)  0.021250    1.000000
3          (beverages)              (nan)  0.016750    0.992593
4       (bottled beer)              (nan)  0.045250    1.000000
..                 ...                ...       ...         ...
81   (rolls/buns, nan)       (whole milk)  0.013500    0.120805
82        (rolls/buns)  (whole milk, nan)  0.013500    0.120670
83  (whole milk, soda)              (nan)  0.011375    1.000000
84         (soda, nan)       (whole milk)  0.011375    0.121495
85              (soda)  (whole milk, nan)  0.011375    0.121495

[86 rows x 4 columns]
```

In [ ]:
```
def count_rules(min_support, min_confidence):
    frequent_itemsets = apriori(df_encoded, min_support=min_support, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)

    return len(rules),rules

#  the range of min_support values and min_confidence thresholds given
min_support_values = [0.001, 0.005, 0.01, 0.05]
min_confidence_thresholds = [0.05, 0.075, 0.1]
assoc_rules = pd.DataFrame(index=min_support_values, columns=min_confidence_thresholds)
results = pd.DataFrame(index=min_support_values, columns=min_confidence_thresholds)

# Iterate over min_support values and min_confidence thresholds to count the number of rules
for min_support in min_support_values:
    for min_confidence in min_confidence_thresholds:
        num_rules,rules = count_rules(min_support, min_confidence)
        results.at[min_support, min_confidence] = num_rules
        assoc_rules.at[min_support, min_confidence] = rules
```

```
# References:
# - Pandas Documentation: https://pandas.pydata.org/pandas-docs/stable/
# - mlxtend Documentation: http://rasbt.github.io/mlxtend/
```

In [ ]:
```
# reffered from https://seaborn.pydata.org/generated/seaborn.heatmap.html
results = results.apply(pd.to_numeric)

plt.figure(figsize=(10, 6))
sns.heatmap(results, annot=True, cmap="YlGnBu", fmt='g', cbar_kws={'label': 'Number of Rules'})
plt.title('Number of Association Rules for Different Support and Confidence Thresholds')
plt.xlabel('Minimum Confidence')
plt.ylabel('Minimum Support')

plt.show()
```



In [ ]:
```
min_support = 0.005
frequent_itemsets = apriori(df_encoded, min_support=min_support, use_colnames=True)

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.0)

filtered_rules = rules[rules['support'] >= min_support]

max_confidence_rule = filtered_rules[filtered_rules['confidence'] == filtered_rules['confidence'].max()]
```

```
print("Association Rule with the Highest Confidence and Minimum Support 0.005:")
print(max_confidence_rule[['antecedents', 'consequents', 'support', 'confidence']])


# References:
# - Pandas Documentation: https://pandas.pydata.org/pandas-docs/stable/
# - mlxtend Documentation: http://rasbt.github.io/mlxtend/
```

```
Association Rule with the Highest Confidence and Minimum Support 0.005:
                     antecedents consequents   support  confidence
1                      (UHT-milk)       (nan)  0.020875         1.0
2                  (baking powder)       (nan)  0.008500         1.0
5                          (beef)       (nan)  0.034250         1.0
7                       (berries)       (nan)  0.021250         1.0
10                  (bottled beer)       (nan)  0.045250         1.0
..                             ...         ...       ...         ...
466            (whole milk, soda)       (nan)  0.011375         1.0
472                (yogurt, soda)       (nan)  0.006625         1.0
479  (whole milk, tropical fruit)       (nan)  0.008125         1.0
485      (yogurt, tropical fruit)       (nan)  0.006000         1.0
490          (whole milk, yogurt)       (nan)  0.009875         1.0

[113 rows x 4 columns]
```

The confidence value is 1

Question 2

In [ ]:

```
import os
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# label map and data directory
labmap = {0: "n02089078-black-and-tan_coonhound",
          1: "n02091831-Saluki",
          2: "n02092002-Scottish_deerhound",
          3: "n02095314-wire-haired_fox_terrier"}
```

```
data_directory = '../DataSet/ProcessedDatasets/'

# parameters
batch_size = 8
image_size = (331, 331)
epochs = 20

# an ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2  # 80% training, 20% validation
)

# data generators
train_generator = datagen.flow_from_directory(
    data_directory,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    classes=list(labmap.values()),
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    data_directory,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    classes=list(labmap.values()),
    subset='validation'
)


# References:
# - TensorFlow Documentation: https://www.tensorflow.org/
# - Keras Documentation: https://keras.io/
# - Matplotlib Documentation: https://matplotlib.org/
# - ImageDataGenerator Documentation: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDa
# - Python os Module Documentation: https://docs.python.org/3/library/os.html
```

```
Found 600 images belonging to 4 classes.
Found 148 images belonging to 4 classes.
```

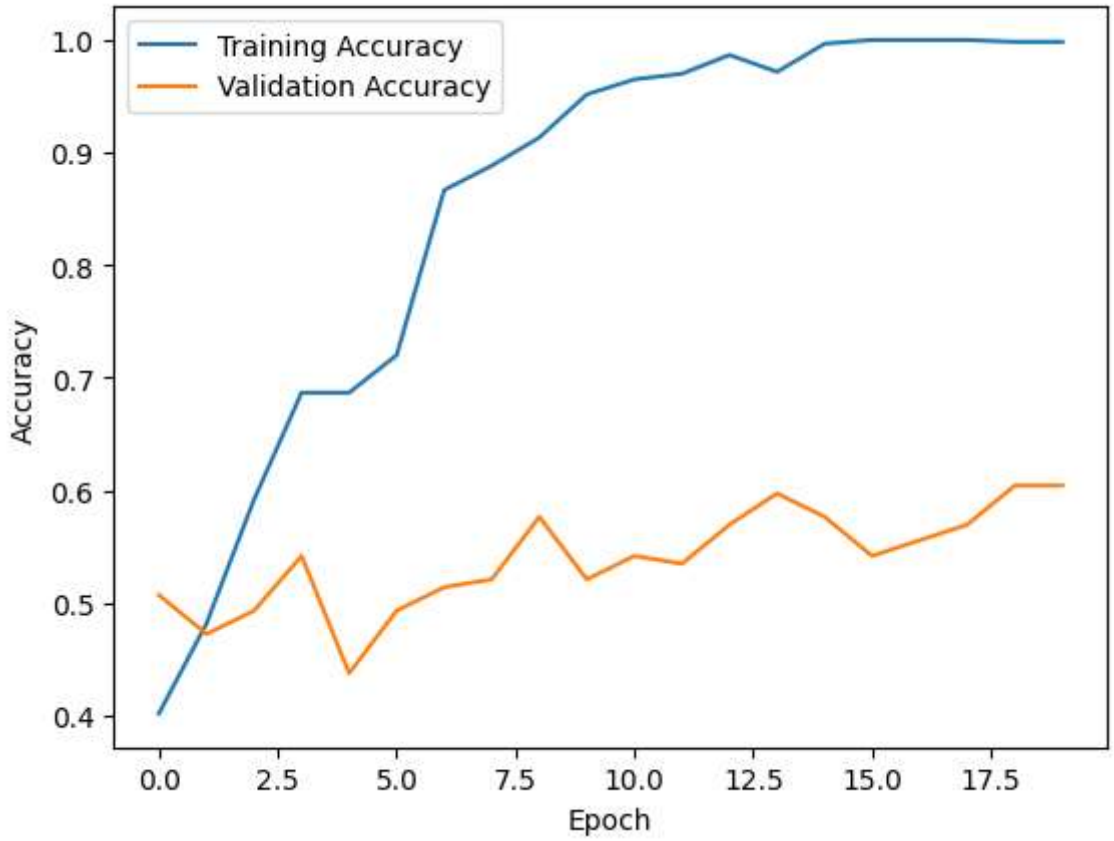
3,3 Filter Size

In [ ]:
```
model0 = models.Sequential()

# Convolutional Layer with 8 3 × 3 filters
model0.add(layers.Conv2D(8, (3, 3), activation='relu', input_shape=(331, 331, 3)))

# 1 max pooling with 2 × 2 pool size
model0.add(layers.MaxPooling2D((2, 2)))

# Flatten the Tensor
model0.add(layers.Flatten())

# 1 hidden layer with 16 nodes for a fully connected neural network
model0.add(layers.Dense(16, activation='relu'))

# Output layer has 4 nodes (since 4 classes) using ‘softmax’ activation function
model0.add(layers.Dense(4, activation='softmax'))

# Compile the model
model0.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Train the model
history = model0.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size
)

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
# References:
# - TensorFlow Documentation: https://www.tensorflow.org/
# - Keras Documentation: https://keras.io/
# - Matplotlib Documentation: https://matplotlib.org/
```

```
WARNING:tensorflow:From c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\backend.p
y:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\layers\poo
ling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\optimizers
\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/20
WARNING:tensorflow:From c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\utils\tf_u
tils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue ins
tead.

WARNING:tensorflow:From c:\Users\kaasa\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\engine\bas
e_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executin
g_eagerly_outside_functions instead.

75/75 [==============================] - 3s 39ms/step - loss: 2.9650 - accuracy: 0.4017 - val_loss: 1.1885 - val_accu
racy: 0.5069
Epoch 2/20
75/75 [==============================] - 3s 37ms/step - loss: 1.1231 - accuracy: 0.4817 - val_loss: 1.1454 - val_accu
racy: 0.4722
Epoch 3/20
75/75 [==============================] - 3s 37ms/step - loss: 0.9529 - accuracy: 0.5917 - val_loss: 1.1178 - val_accu
racy: 0.4931
Epoch 4/20
75/75 [==============================] - 3s 37ms/step - loss: 0.7962 - accuracy: 0.6867 - val_loss: 1.1087 - val_accu
racy: 0.5417
Epoch 5/20
75/75 [==============================] - 3s 37ms/step - loss: 0.7365 - accuracy: 0.6867 - val_loss: 1.3236 - val_accu
racy: 0.4375
Epoch 6/20
75/75 [==============================] - 3s 37ms/step - loss: 0.6890 - accuracy: 0.7200 - val_loss: 1.1400 - val_accu
racy: 0.4931
Epoch 7/20
75/75 [==============================] - 3s 37ms/step - loss: 0.4758 - accuracy: 0.8667 - val_loss: 1.1906 - val_accu
racy: 0.5139
Epoch 8/20
75/75 [==============================] - 3s 37ms/step - loss: 0.4023 - accuracy: 0.8883 - val_loss: 1.2752 - val_accu
racy: 0.5208
Epoch 9/20
```

```
75/75 [==============================] - 3s 37ms/step - loss: 0.3609 - accuracy: 0.9133 - val_loss: 1.1574 - val_accu
racy: 0.5764
Epoch 10/20
75/75 [==============================] - 3s 36ms/step - loss: 0.2729 - accuracy: 0.9517 - val_loss: 1.5291 - val_accu
racy: 0.5208
Epoch 11/20
75/75 [==============================] - 3s 37ms/step - loss: 0.2282 - accuracy: 0.9650 - val_loss: 1.4210 - val_accu
racy: 0.5417
Epoch 12/20
75/75 [==============================] - 3s 37ms/step - loss: 0.1907 - accuracy: 0.9700 - val_loss: 1.5635 - val_accu
racy: 0.5347
Epoch 13/20
75/75 [==============================] - 3s 36ms/step - loss: 0.1409 - accuracy: 0.9867 - val_loss: 1.5129 - val_accu
racy: 0.5694
Epoch 14/20
75/75 [==============================] - 3s 39ms/step - loss: 0.1542 - accuracy: 0.9717 - val_loss: 1.2778 - val_accu
racy: 0.5972
Epoch 15/20
75/75 [==============================] - 3s 37ms/step - loss: 0.1052 - accuracy: 0.9967 - val_loss: 1.5093 - val_accu
racy: 0.5764
Epoch 16/20
75/75 [==============================] - 3s 38ms/step - loss: 0.0697 - accuracy: 1.0000 - val_loss: 1.8391 - val_accu
racy: 0.5417
Epoch 17/20
75/75 [==============================] - 3s 39ms/step - loss: 0.0560 - accuracy: 1.0000 - val_loss: 1.7517 - val_accu
racy: 0.5556
Epoch 18/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0470 - accuracy: 1.0000 - val_loss: 1.8102 - val_accu
racy: 0.5694
Epoch 19/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0474 - accuracy: 0.9983 - val_loss: 1.8246 - val_accu
racy: 0.6042
Epoch 20/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0466 - accuracy: 0.9983 - val_loss: 1.8272 - val_accu
racy: 0.6042
```



Rowan Banner ID: 916459193

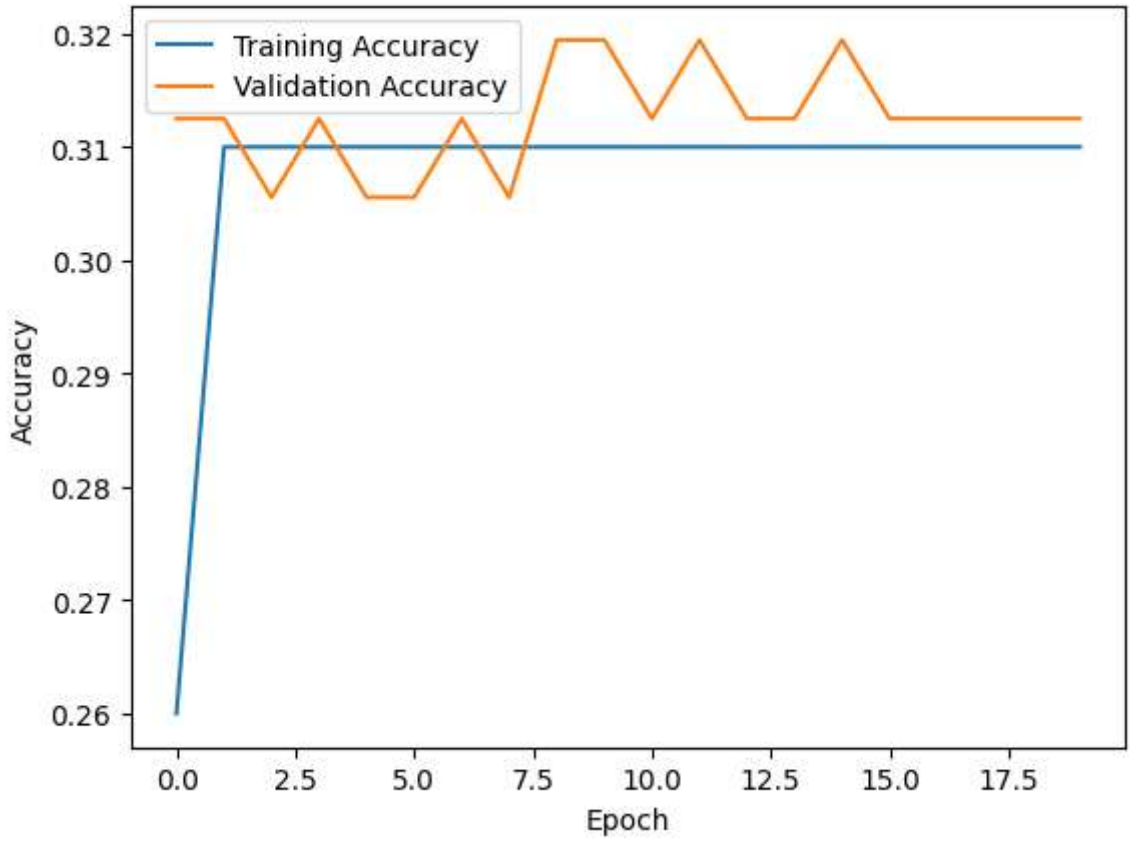
(a) 5*5 Filter Size

In [ ]:
```
model1 = models.Sequential()

# Convolutional Layer with 8 5 × 5 filters
model1.add(layers.Conv2D(8, (5, 5), activation='relu', input_shape=(331, 331, 3)))

# 1 max pooling with 2 × 2 pool size
model1.add(layers.MaxPooling2D((2, 2)))

# Flatten the Tensor
model1.add(layers.Flatten())

# 1 hidden layer with 16 nodes for a fully connected neural network
```

```
model1.add(layers.Dense(16, activation='relu'))

# Output layer has 4 nodes (since 4 classes) using ‘softmax’ activation function
model1.add(layers.Dense(4, activation='softmax'))

# Compile the model
model1.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Train the model
history = model1.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size
)

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# References:
# - TensorFlow Documentation: https://www.tensorflow.org/
# - Keras Documentation: https://keras.io/
# - Matplotlib Documentation: https://matplotlib.org/
```

```
Epoch 1/20
75/75 [==============================] - 3s 37ms/step - loss: 1.7335 - accuracy: 0.2600 - val_loss: 1.3847 - val_accu
racy: 0.3125
Epoch 2/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3841 - accuracy: 0.3100 - val_loss: 1.3828 - val_accu
racy: 0.3125
Epoch 3/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3820 - accuracy: 0.3100 - val_loss: 1.3808 - val_accu
racy: 0.3056
Epoch 4/20
75/75 [==============================] - 3s 35ms/step - loss: 1.3801 - accuracy: 0.3100 - val_loss: 1.3784 - val_accu
racy: 0.3125
Epoch 5/20
75/75 [==============================] - 3s 35ms/step - loss: 1.3786 - accuracy: 0.3100 - val_loss: 1.3776 - val_accu
racy: 0.3056
Epoch 6/20
75/75 [==============================] - 3s 35ms/step - loss: 1.3776 - accuracy: 0.3100 - val_loss: 1.3779 - val_accu
racy: 0.3056
Epoch 7/20
75/75 [==============================] - 3s 35ms/step - loss: 1.3765 - accuracy: 0.3100 - val_loss: 1.3741 - val_accu
racy: 0.3125
Epoch 8/20
75/75 [==============================] - 3s 35ms/step - loss: 1.3758 - accuracy: 0.3100 - val_loss: 1.3758 - val_accu
racy: 0.3056
Epoch 9/20
75/75 [==============================] - 3s 35ms/step - loss: 1.3753 - accuracy: 0.3100 - val_loss: 1.3712 - val_accu
racy: 0.3194
Epoch 10/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3748 - accuracy: 0.3100 - val_loss: 1.3723 - val_accu
racy: 0.3194
Epoch 11/20
75/75 [==============================] - 3s 35ms/step - loss: 1.3744 - accuracy: 0.3100 - val_loss: 1.3737 - val_accu
racy: 0.3125
Epoch 12/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3740 - accuracy: 0.3100 - val_loss: 1.3704 - val_accu
racy: 0.3194
Epoch 13/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3739 - accuracy: 0.3100 - val_loss: 1.3721 - val_accu
racy: 0.3125
Epoch 14/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3737 - accuracy: 0.3100 - val_loss: 1.3706 - val_accu
racy: 0.3125
```

```
Epoch 15/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3735 - accuracy: 0.3100 - val_loss: 1.3696 - val_accu
racy: 0.3194
Epoch 16/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3735 - accuracy: 0.3100 - val_loss: 1.3730 - val_accu
racy: 0.3125
Epoch 17/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3733 - accuracy: 0.3100 - val_loss: 1.3728 - val_accu
racy: 0.3125
Epoch 18/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3733 - accuracy: 0.3100 - val_loss: 1.3714 - val_accu
racy: 0.3125
Epoch 19/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3732 - accuracy: 0.3100 - val_loss: 1.3714 - val_accu
racy: 0.3125
Epoch 20/20
75/75 [==============================] - 3s 36ms/step - loss: 1.3733 - accuracy: 0.3100 - val_loss: 1.3712 - val_accu
racy: 0.3125
```



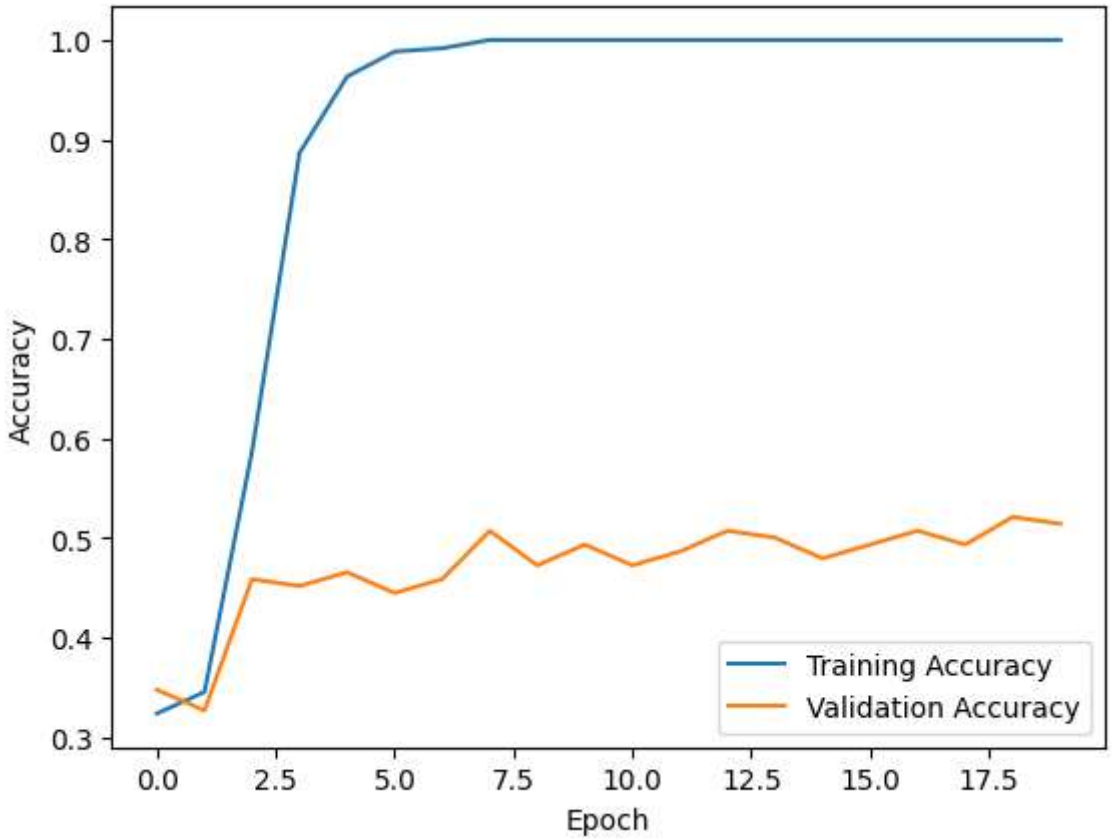
7*7 Filter Size

In [ ]:
```
model2 = models.Sequential()

# Convolutional Layer with 8 7 × 7 filters
model2.add(layers.Conv2D(8, (7, 7), activation='relu', input_shape=(331, 331, 3)))

# 1 max pooling with 2 × 2 pool size
model2.add(layers.MaxPooling2D((2, 2)))

# Flatten the Tensor
model2.add(layers.Flatten())

# 1 hidden layer with 16 nodes for a fully connected neural network
model2.add(layers.Dense(16, activation='relu'))
```

```
# Output layer has 4 nodes (since 4 classes) using 'softmax' activation function
model2.add(layers.Dense(4, activation='softmax'))

# Compile the model
model2.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Train the model
history = model2.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size
)

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


# References:
# - TensorFlow Documentation: https://www.tensorflow.org/
# - Keras Documentation: https://keras.io/
# - Matplotlib Documentation: https://matplotlib.org/
```

```
Epoch 1/20
75/75 [==============================] - 3s 39ms/step - loss: 2.5722 - accuracy: 0.3233 - val_loss: 1.3670 - val_accu
racy: 0.3472
Epoch 2/20
75/75 [==============================] - 3s 37ms/step - loss: 1.3030 - accuracy: 0.3450 - val_loss: 1.3291 - val_accu
racy: 0.3264
Epoch 3/20
75/75 [==============================] - 3s 38ms/step - loss: 0.9599 - accuracy: 0.5867 - val_loss: 1.3747 - val_accu
racy: 0.4583
Epoch 4/20
75/75 [==============================] - 3s 38ms/step - loss: 0.4348 - accuracy: 0.8867 - val_loss: 1.4244 - val_accu
racy: 0.4514
Epoch 5/20
75/75 [==============================] - 3s 37ms/step - loss: 0.1729 - accuracy: 0.9633 - val_loss: 2.1354 - val_accu
racy: 0.4653
Epoch 6/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0601 - accuracy: 0.9883 - val_loss: 2.1604 - val_accu
racy: 0.4444
Epoch 7/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0324 - accuracy: 0.9917 - val_loss: 2.6269 - val_accu
racy: 0.4583
Epoch 8/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0211 - accuracy: 1.0000 - val_loss: 2.6990 - val_accu
racy: 0.5069
Epoch 9/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0075 - accuracy: 1.0000 - val_loss: 2.6248 - val_accu
racy: 0.4722
Epoch 10/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0037 - accuracy: 1.0000 - val_loss: 2.5683 - val_accu
racy: 0.4931
Epoch 11/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 2.8576 - val_accu
racy: 0.4722
Epoch 12/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 2.7529 - val_accu
racy: 0.4861
Epoch 13/20
75/75 [==============================] - 3s 38ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 2.7974 - val_accu
racy: 0.5069
Epoch 14/20
75/75 [==============================] - 3s 38ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 3.0074 - val_accu
racy: 0.5000
```

```
Epoch 15/20
75/75 [==============================] - 3s 37ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 3.0727 - val_accu
racy: 0.4792
Epoch 16/20
75/75 [==============================] - 3s 37ms/step - loss: 8.6346e-04 - accuracy: 1.0000 - val_loss: 3.1099 - val_
accuracy: 0.4931
Epoch 17/20
75/75 [==============================] - 3s 37ms/step - loss: 7.3399e-04 - accuracy: 1.0000 - val_loss: 3.1448 - val_
accuracy: 0.5069
Epoch 18/20
75/75 [==============================] - 3s 37ms/step - loss: 6.1140e-04 - accuracy: 1.0000 - val_loss: 3.2050 - val_
accuracy: 0.4931
Epoch 19/20
75/75 [==============================] - 3s 38ms/step - loss: 5.0768e-04 - accuracy: 1.0000 - val_loss: 3.1439 - val_
accuracy: 0.5208
Epoch 20/20
75/75 [==============================] - 3s 37ms/step - loss: 4.2353e-04 - accuracy: 1.0000 - val_loss: 3.3316 - val_
accuracy: 0.5139
```



Observation

The first model is learning slowly where the training and validation loss at initial epochs is very low, which gradually increased.
The first model with a 3x3 filter is close to just right with the training and validation accuracies of (99.83,60.42) respectively

The first model with a 5x5 filter is underfit with the training and validation accuracies of (31.00,31.29) respectively

The first model with a 7x7 filter is overfit with the training and validation accuracies of (100,51.39) respectively

Model can be overfit when the training accuracy is very high and the validation accuracy is very low.

References:

- Pandas Documentation: https://pandas.pydata.org/pandas-docs/stable/
- mlxtend Documentation: http://rasbt.github.io/mlxtend/
- TensorFlow Documentation: https://www.tensorflow.org/
- Keras Documentation: https://keras.io/
- Matplotlib Documentation: https://matplotlib.org/
- ImageDataGenerator Documentation: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator