In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
import os
import cv2
import matplotlib.pyplot as plt
from PIL import Image
from skimage import io
from google.colab.patches import cv2_imshow
import numpy as np
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
import math
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from numpy import array
from keras.utils import np_utils
from keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, Ba
import matplotlib.pyplot as plt
```

1. Association Rule Generation from Transaction Data

In [3]:
```
dataset = pd.read_csv('/content/drive/MyDrive/Grocery_Items_36.csv')
dataset.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9
0	meat	pastry	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	tropical fruit	other vegetables	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	soda	herbs	canned beer	soda	seasonal products	dishes	yogurt	mayonnaise	flower (seeds)	NaN
3	hard cheese	salty snack	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	frankfurter	brown bread	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

(c) Using minimum support = 0.01 and minimum confidence threshold = 0.1, what are the association rules you can extract from your dataset? (0.5 point) (see http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association

(http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/)

In [4]:
```
for i in range(11):

    dataset[f"{i}"].fillna("NULL", inplace = True)

dataset.shape
```

Out[4]: (8000, 11)

In [5]:
```
dataset_List = dataset.values.tolist()
```

In [22]:
```
te = TransactionEncoder()
te_ary = te.fit(dataset_List).transform(dataset_List)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

Out[22]:

	Instant food products	NULL	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	bags	baking powder	bathroom cleaner
0	False	True	False	False	False	False	False	False	False
1	False	True	False	False	False	False	False	False	False
2	False	True	False	False	False	False	False	False	False
3	False	True	False	False	False	False	False	False	False
4	False	True	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
7995	False	True	False	False	False	False	False	False	False
7996	False	True	False	False	False	False	False	False	False
7997	False	True	False	False	False	False	False	False	False
7998	False	True	False	False	False	False	False	False	False
7999	False	True	False	False	False	False	False	False	False

8000 rows × 167 columns

Reference: http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/ (http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/)

In [23]:

```
#going to drop null column
#Because NULL is not an item

df.drop('NULL', inplace= True, axis= 1)
df.shape
```

Out[23]: (8000, 166)

In [24]:

```
#freequent items dataset

from mlxtend.frequent_patterns import apriori

#given condition minimum support : 0.01

frequent_items = apriori(df, min_support=0.01, use_colnames=True)

frequent_items['length'] = frequent_items['itemsets'].apply(lambda x: le

frequent_items
```

Out[24]:

	support	itemsets	length
0	0.022250	(UHT-milk)	1
1	0.036375	(beef)	1
2	0.021000	(berries)	1
3	0.015375	(beverages)	1
4	0.046375	(bottled beer)	1
...	...	...	...
64	0.011375	(other vegetables, rolls/buns)	2
65	0.014250	(whole milk, other vegetables)	2
66	0.015000	(whole milk, rolls/buns)	2
67	0.011250	(whole milk, soda)	2
68	0.011875	(whole milk, yogurt)	2

69 rows × 3 columns

In [25]:

```
from mlxtend.frequent_patterns import association_rules

#given conditions Minimum confidence : 0.1

assn_rules = association_rules(frequent_items, metric="confidence", min_

print(f'We got {assn_rules.shape[0]} association rules for given msv = 0
assn_rules
```

```
We got 5 association rules for given msv = 0.01 and mct = 0.1
```

Out[25]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	l
0	(rolls/buns)	(other vegetables)	0.109875	0.121875	0.011375	0.103527	0.849450	-0
1	(other vegetables)	(whole milk)	0.121875	0.160125	0.014250	0.116923	0.730199	-0
2	(rolls/buns)	(whole milk)	0.109875	0.160125	0.015000	0.136519	0.852576	-0
3	(soda)	(whole milk)	0.096375	0.160125	0.011250	0.116732	0.729002	-0
4	(yogurt)	(whole milk)	0.085000	0.160125	0.011875	0.139706	0.872480	-0

In [26]:

```
assn_rules.shape
```

Out[26]: (5, 9)

Use minimum support values (msv): 0.001, 0.005, 0.01, 0.05 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 (https://seaborn)](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. (2.5 points)

In [27]:

```
msv = [0.001, 0.005, 0.01, 0.05] #support values
mct = [0.05, 0.075, 0.1] #confidence values
rows = []

for i in msv:
    frequent_items = apriori(df, min_support=i, use_colnames=True)
    for j in mct:
        assn_rules_d = association_rules(frequent_items, metric="confide
        rows.append([i, j, assn_rules_d.shape[0]])

#df_counts = pd.DataFrame(counts, columns = msv, index = mct)
```

In [28]:
```
rows
```

Out[28]:
```
[[0.001, 0.05, 502],
 [0.001, 0.075, 293],
 [0.001, 0.1, 151],
 [0.005, 0.05, 54],
 [0.005, 0.075, 39],
 [0.005, 0.1, 25],
 [0.01, 0.05, 10],
 [0.01, 0.075, 8],
 [0.01, 0.1, 5],
 [0.05, 0.05, 0],
 [0.05, 0.075, 0],
 [0.05, 0.1, 0]]
```

In [29]:
```
df_matrix = pd.DataFrame(rows, columns = ["Support", "Confidence", "Coun
df_matrix
```

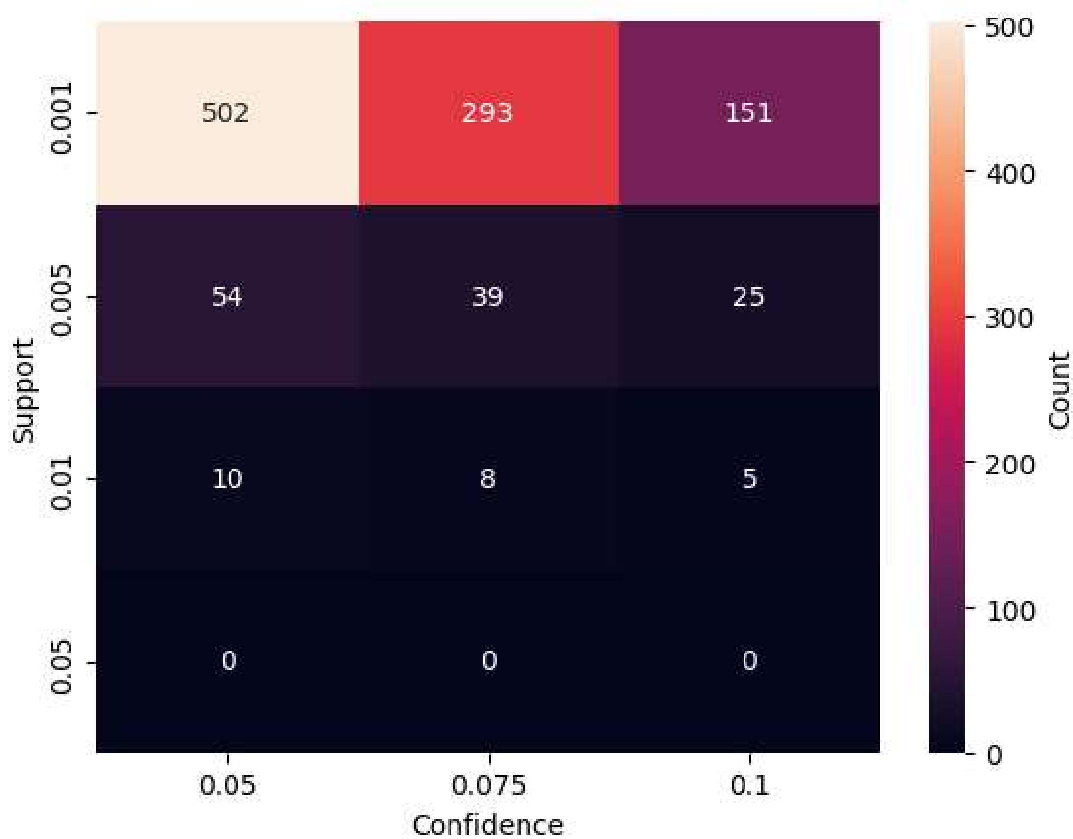
Out[29]:

	Support	Confidence	Count
0	0.001	0.050	502
1	0.001	0.075	293
2	0.001	0.100	151
3	0.005	0.050	54
4	0.005	0.075	39
5	0.005	0.100	25
6	0.010	0.050	10
7	0.010	0.075	8
8	0.010	0.100	5
9	0.050	0.050	0
10	0.050	0.075	0
11	0.050	0.100	0

In [30]:
```
sns.heatmap(data=df_matrix.pivot('Support', 'Confidence', 'Count'), anno
```

```
<ipython-input-30-cdb571ffce6f>:1: FutureWarning: In a future version o
f pandas all arguments of DataFrame.pivot will be keyword-only.
  sns.heatmap(data=df_matrix.pivot('Support', 'Confidence', 'Count'), a
nnot=True, fmt='.0f', cbar_kws={'label': 'Count'})
```

Out[30]:
```
<Axes: xlabel='Confidence', ylabel='Support'>
```



List the association rule(s) (i.e., one or more rules depending on your dataset) that have the highest confidence for minimum support = 0.005. What is that confidence value? (1 point) 1

In [31]:
```
frequent_items = apriori(df, min_support=0.005, use_colnames=True)
```

In [32]:

```
assn_rules_e = association_rules(frequent_items, metric = 'confidence', 

print(f'We got {assn_rules_e.shape[0]} association rules')

assn_rules_e
```

```
We got 25 association rules
```

Out[32]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(beef)	(whole milk)	0.036375	0.160125	0.005500	0.151203	0.944279
1	(bottled beer)	(whole milk)	0.046375	0.160125	0.007250	0.156334	0.976326
2	(bottled water)	(whole milk)	0.059375	0.160125	0.008000	0.134737	0.841448
3	(canned beer)	(rolls/buns)	0.048125	0.109875	0.005000	0.103896	0.945585
4	(canned beer)	(whole milk)	0.048125	0.160125	0.006000	0.124675	0.778612
5	(citrus fruit)	(whole milk)	0.053000	0.160125	0.007250	0.136792	0.854285
6	(frankfurter)	(other vegetables)	0.038750	0.121875	0.005500	0.141935	1.164599
7	(frankfurter)	(whole milk)	0.038750	0.160125	0.005375	0.138710	0.866259
8	(newspapers)	(whole milk)	0.038625	0.160125	0.005625	0.145631	0.909484
9	(pip fruit)	(other vegetables)	0.049875	0.121875	0.005625	0.112782	0.925390
10	(rolls/buns)	(other vegetables)	0.109875	0.121875	0.011375	0.103527	0.849450
11	(sausage)	(other vegetables)	0.057875	0.121875	0.005875	0.101512	0.832918
12	(whipped/sour cream)	(other vegetables)	0.043625	0.121875	0.005000	0.114613	0.940416
13	(other vegetables)	(whole milk)	0.121875	0.160125	0.014250	0.116923	0.730199
14	(pastry)	(whole milk)	0.053250	0.160125	0.007750	0.145540	0.908914
15	(pip fruit)	(rolls/buns)	0.049875	0.109875	0.005250	0.105263	0.958026
16	(pip fruit)	(whole milk)	0.049875	0.160125	0.007500	0.150376	0.939116
17	(rolls/buns)	(whole milk)	0.109875	0.160125	0.015000	0.136519	0.852576
18	(root vegetables)	(whole milk)	0.067750	0.160125	0.007375	0.108856	0.679819
19	(sausage)	(soda)	0.057875	0.096375	0.006250	0.107991	1.120533
20	(sausage)	(whole milk)	0.057875	0.160125	0.008000	0.138229	0.863256
21	(shopping bags)	(whole milk)	0.045625	0.160125	0.005625	0.123288	0.769946
22	(soda)	(whole milk)	0.096375	0.160125	0.011250	0.116732	0.729002
23	(tropical fruit)	(whole milk)	0.064750	0.160125	0.008500	0.131274	0.819823
24	(yogurt)	(whole milk)	0.085000	0.160125	0.011875	0.139706	0.872480

The association rule with highest confidence value 0.156334 and there is only one association rule with this value

In [33]:
```
assn_rules_e[(assn_rules_e['confidence'] == assn_rules_e['confidence'].ma
```

Out[33]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	le
1	(bottled beer)	(whole milk)	0.046375	0.160125	0.00725	0.156334	0.976326	-0.(

2. Image Classification using CNN

In [34]:
```
DF=pd.read_csv('/content/drive/MyDrive/DM P1/Weed-4class-36/Weed-4class-3
DF
```

Out[34]:

	Filename	Label	Species
0	20161207-112417-0.jpg	8	Negative
1	20161207-112431-0.jpg	8	Negative
2	20161207-112802-0.jpg	8	Negative
3	20161207-112812-0.jpg	8	Negative
4	20170128-101909-0.jpg	8	Negative
...	...	...	...
13277	20171025-172145-3.jpg	3	Parthenium
13278	20171025-172200-3.jpg	3	Parthenium
13279	20171025-172226-3.jpg	3	Parthenium
13280	20171025-172236-3.jpg	3	Parthenium
13281	20171025-172247-3.jpg	3	Parthenium

13282 rows × 3 columns

In [35]:
```
DF_part=DF.loc[DF['Species']=='Parthenium']
DF_lant=DF.loc[DF['Species']=='Lantana']
DF_snake=DF.loc[DF['Species']=='Snake weed']
DF_siam=DF.loc[DF['Species']=='Siam weed']

part_imgs=list(DF_part['Filename'])
lant_imgs=list(DF_lant['Filename'])
snake_imgs=list(DF_snake['Filename'])
siam_imgs=list(DF_siam['Filename'])
```

In [38]:

```
DF_snake
```

Out[38]:

	Filename	Label	Species
10170	20161207-142702-0.jpg	7	Snake weed
10171	20161207-142721-0.jpg	7	Snake weed
10172	20161207-143329-0.jpg	7	Snake weed
10173	20161207-143344-0.jpg	7	Snake weed
10174	20161207-144425-0.jpg	7	Snake weed
...	...	...	...
11181	20180109-092127-2.jpg	7	Snake weed
11182	20180109-092137-2.jpg	7	Snake weed
11183	20180109-094515-2.jpg	7	Snake weed
11184	20180109-100055-2.jpg	7	Snake weed
11185	20180109-104330-2.jpg	7	Snake weed

1016 rows × 3 columns

In [39]:

```
part=[]
lant=[]
snake=[]

labels_part=[]
labels_lant=[]
labels_snake=[]

for img in part_imgs:
    img_read = plt.imread( '/content/drive/MyDrive/DM P1/Weed-4class-36/
    img_resize = cv2.resize(img_read, (64,64))
    img_array = np.asarray(img_resize,dtype=np.float32)
    part.append(img_array)
    labels_part.append(3)

for img in lant_imgs:
    img_read = plt.imread( '/content/drive/MyDrive/DM P1/Weed-4class-36/
    img_resize = cv2.resize(img_read, (64,64))
    img_array = np.asarray(img_resize,dtype=np.float32)
    lant.append(img_array)
    labels_lant.append(1)

for img in snake_imgs:
    img_read = plt.imread( '/content/drive/MyDrive/DM P1/Weed-4class-36/
    img_resize = cv2.resize(img_read, (64,64))
    img_array = np.asarray(img_resize,dtype=np.float32)
    snake.append(img_array)
    labels_snake.append(7)
```

In [40]:

```
X_part_train, X_part_test, y_part_train, y_part_test = train_test_split(
X_lant_train, X_lant_test, y_lant_train, y_lant_test = train_test_split(
X_snake_train, X_snake_test, y_snake_train, y_snake_test = train_test_sp
```

In [42]:

```
X_train=X_part_train+X_lant_train+X_snake_train
y_train=y_part_train+y_lant_train+y_snake_train

X_test=X_part_test+X_lant_test+X_snake_test
y_test=y_part_test+y_lant_test+y_snake_test

X_train=np.array(X_train)
y_train=np.array(y_train)
X_test=np.array(X_test)
y_test=np.array(y_test)

sort_y_train = np.searchsorted([1,3,7],  y_train)
sort_y_test = np.searchsorted([1,3,7],  y_test)

y_train = np_utils.to_categorical(sort_y_train)
y_test = np_utils.to_categorical(sort_y_test)
```

In [43]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(2480, 64, 64, 3)
(2480, 3)
(622, 64, 64, 3)
(622, 3)
```

2. [Image Classification using CNN] Construct a 3-class classification (ignore negative class and 1 other weed class) 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 iii Flatten the Tensor iv 1 hidden layer with 16 nodes for fully connected neural network v Output layer has 3 nodes (since 3 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

In [48]:

```
from tensorflow import keras

# Define the input shape
input_shape = (64,64,3)

# Initialize the model
model = keras.Sequential()
model.add(Conv2D(8, (3, 3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Compile the model with Adam optimizer and categorical cross entropy los
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=

# Train the model for 20 epochs
model_1=model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y
```

```
Epoch 1/20
78/78 [==============================] - 2s 7ms/step - loss: 10.0164 -
accuracy: 0.3399 - val_loss: 1.0985 - val_accuracy: 0.3424
Epoch 2/20
78/78 [==============================] - 0s 5ms/step - loss: 1.1005 - a
ccuracy: 0.3431 - val_loss: 1.0985 - val_accuracy: 0.3424
Epoch 3/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0985 - val_accuracy: 0.3424
Epoch 4/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 5/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 6/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 7/20
78/78 [==============================] - 0s 6ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 8/20
78/78 [==============================] - 1s 6ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 9/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 10/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 11/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 12/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0986 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 13/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 14/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 15/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 16/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 17/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0984 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 18/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0984 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 19/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
```

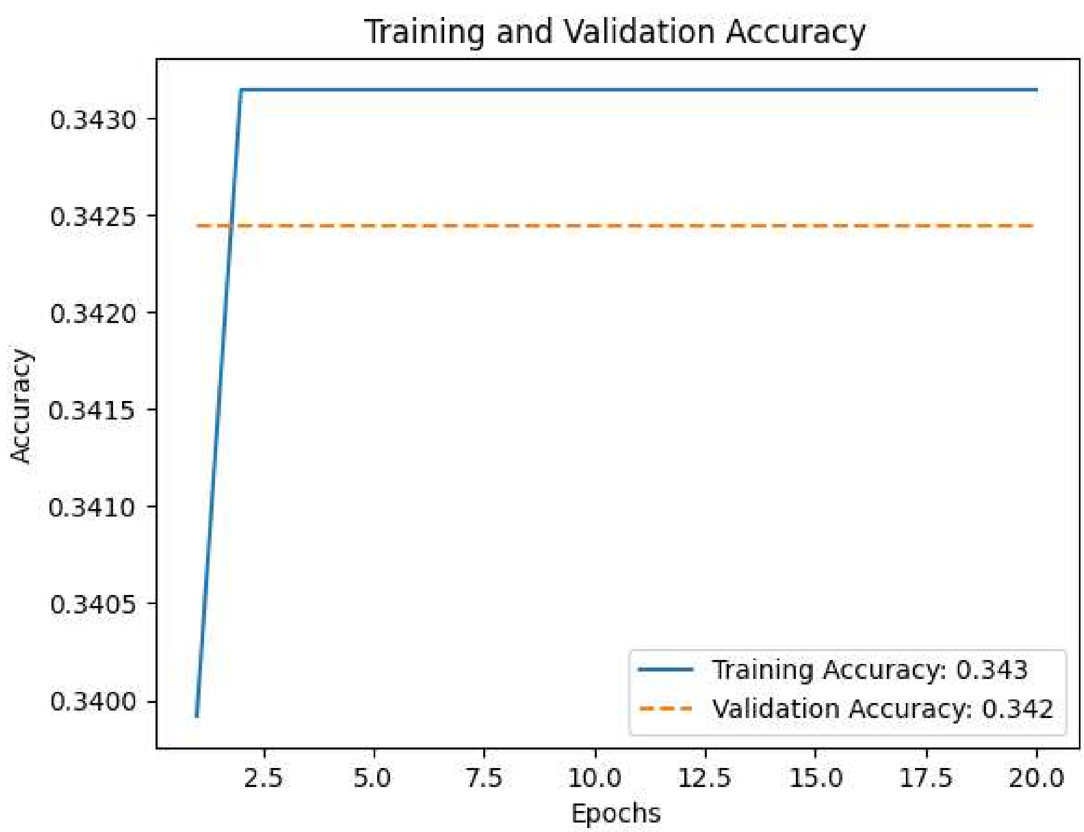
```
Epoch 20/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0984 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
```

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)

In [49]:

```
train_acc = model_1.history['accuracy']
val_acc = model_1.history['val_accuracy']

epochs = range(1, len(train_acc) + 1)
plt.plot(epochs, train_acc, '-', label=f'Training Accuracy: {train_acc[-1
plt.plot(epochs, val_acc, '--', label=f'Validation Accuracy: {val_acc[-1
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

```

```

Do ONE of the following 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 with all other parameters unchanged (b) Train the CNN using 2 other number of filters: 4 and 16 with all other parameters unchanged (c) Train the CNN using 2 other number of nodes in the hidden layer: 8 and 32 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.

Banner ID: 916426214

In [46]:

```
from tensorflow import keras

# Define the input shape
input_shape = (64,64,3)

# Initialize the model
model = keras.Sequential()

model.add(Conv2D(8, (3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(4, (3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(16, (3, 3), activation='relu', input_shape=input_shape)
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())

model.add(Dense(16, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Compile the model with Adam optimizer and categorical cross entropy los
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=

# Train the model for 20 epochs
model_2=model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y
```

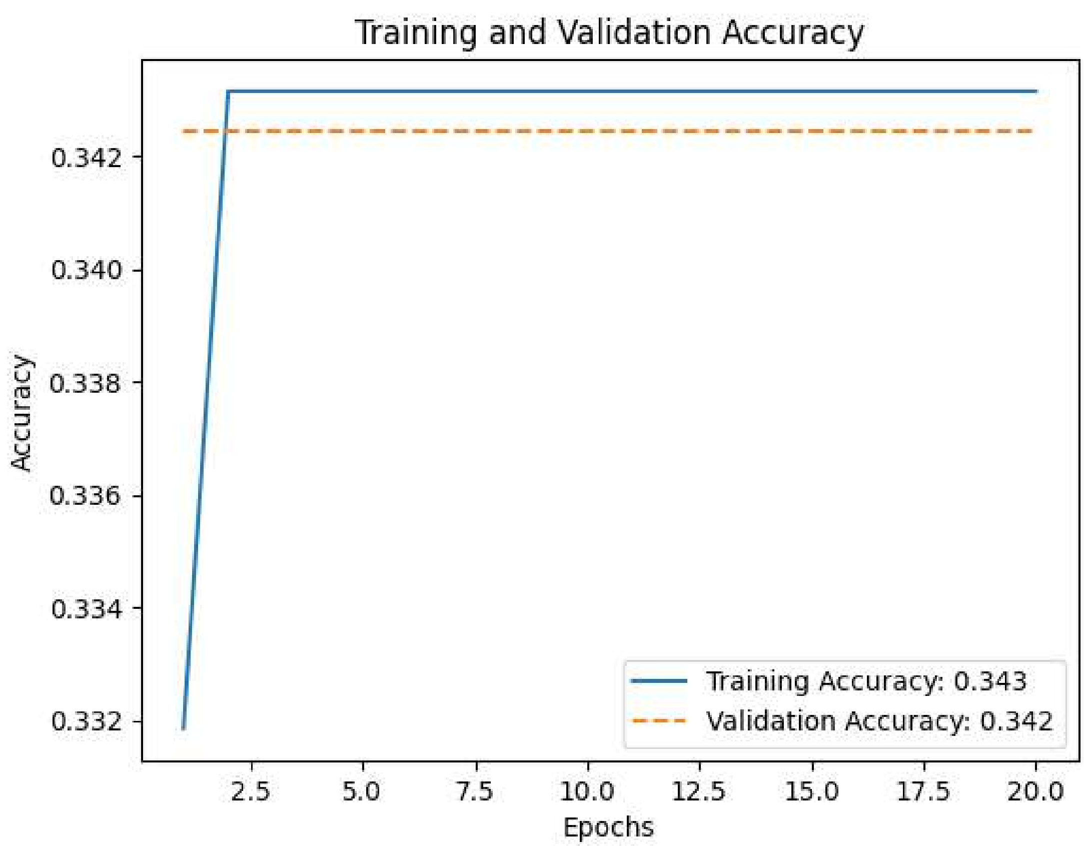
```
Epoch 1/20
78/78 [==============================] - 3s 11ms/step - loss: 14.4490 -
accuracy: 0.3319 - val_loss: 1.0985 - val_accuracy: 0.3424
Epoch 2/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0986 - a
ccuracy: 0.3431 - val_loss: 1.0985 - val_accuracy: 0.3424
Epoch 3/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0985 - val_accuracy: 0.3424
Epoch 4/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0985 - val_accuracy: 0.3424
Epoch 5/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0985 - val_accuracy: 0.3424
Epoch 6/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 7/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 8/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 9/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 10/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 11/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 12/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0984 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 13/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 14/20
78/78 [==============================] - 0s 5ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 15/20
78/78 [==============================] - 0s 6ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 16/20
78/78 [==============================] - 0s 6ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 17/20
78/78 [==============================] - 0s 6ms/step - loss: 1.0985 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 18/20
78/78 [==============================] - 0s 6ms/step - loss: 1.0984 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
Epoch 19/20
78/78 [==============================] - 0s 6ms/step - loss: 1.0984 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
```

```
Epoch 20/20
78/78 [==============================] - 0s 6ms/step - loss: 1.0986 - a
ccuracy: 0.3431 - val_loss: 1.0984 - val_accuracy: 0.3424
```

Plot the learning curves (i.e., x-axis: number of epochs; y-axis: training and validation accuracy - 2 curves) for the above 2 configurations (1 points)

In [47]:
```
train_acc = model_2.history['accuracy']
val_acc = model_2.history['val_accuracy']

epochs = range(1, len(train_acc) + 1)
plt.plot(epochs, train_acc, '-', label=f'Training Accuracy: {train_acc[-1
plt.plot(epochs, val_acc, '--', label=f'Validation Accuracy: {val_acc[-1
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [ ]:

In [ ]:

```
In [ ]:
```

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). Are there model overfit or underfit or just right? (1 point)

Based on the training and validation accuracies in the first model --- The model is neither overfitting nor underfitting but not performing better as both accuracies are very low

Based on the training and validation accuracies in the second model --- The model is neither overfitting nor underfitting but not performing better as both accuracies are very low

Also the addition of layers did not make accuracy better which means did not make model better , as both accuracies are same