In [5]:
```
!pip install pandas
!pip install numpy
```

```
Requirement already satisfied: pandas in c:\users\puspa\anaconda3\lib\site-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\puspa\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\puspa\anaconda3\lib\site-packages (from pandas) (2022.7)
Requirement already satisfied: numpy>=1.21.0 in c:\users\puspa\anaconda3\lib\site-packages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in c:\users\puspa\anaconda3\lib\site-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Requirement already satisfied: numpy in c:\users\puspa\anaconda3\lib\site-packages (1.24.3)
```

In [4]:
```
!pip install mlxtend
```

```
Requirement already satisfied: mlxtend in c:\users\puspa\anaconda3\lib\site-packages (0.23.0)
Requirement already satisfied: scipy>=1.2.1 in c:\users\puspa\anaconda3\lib\site-packages (from mlxtend) (1.10.1)
Requirement already satisfied: numpy>=1.16.2 in c:\users\puspa\anaconda3\lib\site-packages (from mlxtend) (1.24.3)
Requirement already satisfied: pandas>=0.24.2 in c:\users\puspa\anaconda3\lib\site-packages (from mlxtend) (1.5.3)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\puspa\anaconda3\lib\site-packages (from mlxtend) (1.3.0)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\puspa\anaconda3\lib\site-packages (from mlxtend) (3.7.1)
Requirement already satisfied: joblib>=0.13.2 in c:\users\puspa\anaconda3\lib\site-packages (from mlxtend) (1.2.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\puspa\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\puspa\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\puspa\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\puspa\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\puspa\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\puspa\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\puspa\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\puspa\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\puspa\anaconda3\lib\site-packages (from pandas>=0.24.2->mlxtend) (2022.7)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\puspa\anaconda3\lib\site-packages (from scikit-learn>=1.0.2->mlxtend) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\puspa\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [231...
```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in c:\users\puspa\anaconda3\lib\site-packages (2.15.0)
Requirement already satisfied: tensorflow-intel==2.15.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow) (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.0.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (3.7.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes~=0.2.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.24.3)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (3.3.0)
Requirement already satisfied: packaging in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (23.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (4.23.4)
Requirement already satisfied: setuptools in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (68.0.0)
Requirement already satisfied: six>=1.12.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (4.7.1)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (0.31.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (1.59.3)
Requirement already satisfied: tensorboard<2.16,>=2.15 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.15.1)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorflow-intel==2.15.0->tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\puspa\anaconda3\lib\site-packages (from astunparse>=1.6.0->tensorflow-intel==2.15.0->tensorflow) (0.38.4)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\puspa\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.23.4)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in c:\users\puspa\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (1.1.0)
Requirement already satisfied: markdown>=2.6.8 in c:\users\puspa\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (3.4.1)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\users\puspa\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\puspa\anaconda3\lib\site-packages (from tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.2.3)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\puspa\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\puspa\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in c:\users\puspa\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\puspa\anaconda3\lib\site-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\puspa\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\puspa\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\puspa\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\puspa\anaconda3\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2023.7.22)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\puspa\anaconda3\lib\site-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.1.1)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in c:\users\puspa\anaconda3\lib\site-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\puspa\anaconda3\lib\site-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (3.2.2)
```

In [220...
```
#1 [Association Rule Generation from Transaction Data]

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

file_path = r'C:\Users\puspa\OneDrive\Desktop\Grocery_Items_26.csv'
data = pd.read_csv(file_path)
data = data.fillna('missing')
data = data.astype(str)
data.head()
```

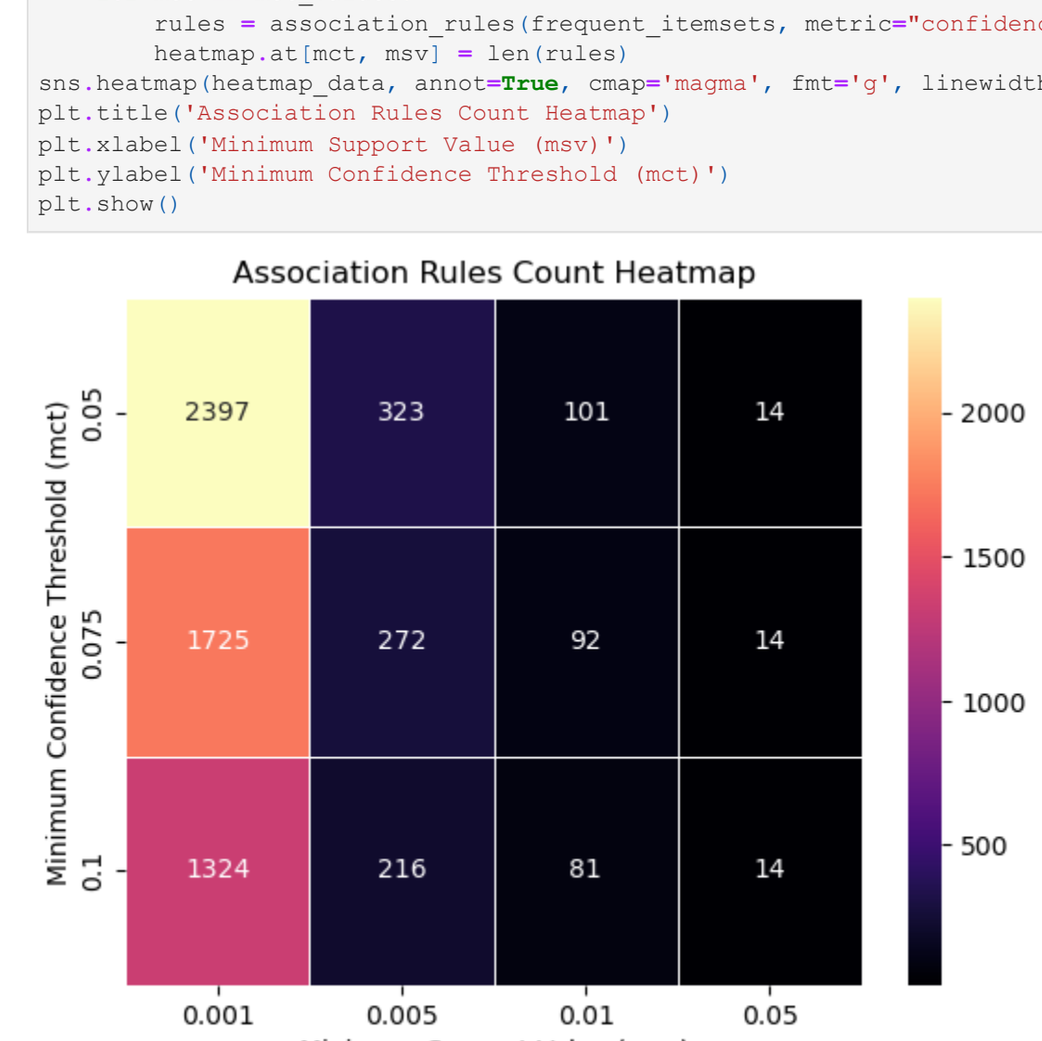
Out[220]:

	0	1	2	3	4	5	6	7	8	9	10
0	chocolate	citrus fruit	bottled beer	bottled water	red/blush wine	missing	missing	missing	missing	missing	missing
1	whole milk	pastry	missing	missing	missing	missing	missing	missing	missing	missing	missing
2	butter milk	sliced cheese	rolls/buns	missing	missing	missing	missing	missing	missing	missing	missing
3	meat	other vegetables	missing	missing	missing	missing	missing	missing	missing	missing	missing
4	dessert	other vegetables	missing	missing	missing	missing	missing	missing	missing	missing	missing

In [237...
```
#Using minimum support = 0.01 and minimum confidence threshold = 0.1, what are the association rules you can extract from your dataset
tran_set = data.apply(lambda x: x.dropna().tolist(), axis=1).tolist()
te= TransactionEncoder()
transactions_encoded = te.fit(tran_set).transform(tran_set)
df_transactions_encoded = pd.DataFrame(transactions_encoded, columns=te.columns_)
frequent_itemsets = apriori(df_transactions_encoded, min_support=0.01, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.1)
print(rules)
```

```
              antecedents            consequents  antecedent support  \
0              (UHT-milk)              (missing)            0.021750
1                  (beef)              (missing)            0.032750
2               (berries)              (missing)            0.021250
3             (beverages)              (missing)            0.016250
4          (bottled beer)              (missing)            0.045375
..                    ...                    ...                 ...
76          (soda, missing)         (whole milk)            0.095375
77                 (soda)  (whole milk, missing)            0.095375
78   (whole milk, yogurt)              (missing)            0.012625
79      (yogurt, missing)           (whole milk)            0.086125
80               (yogurt)  (whole milk, missing)            0.086125

    consequent support   support  confidence      lift  leverage  conviction  \
0               1.0000  0.021750    1.000000  1.000000  0.000000         inf
1               1.0000  0.032750    1.000000  1.000000  0.000000         inf
2               1.0000  0.021250    1.000000  1.000000  0.000000         inf
3               1.0000  0.016250    1.000000  1.000000  0.000000         inf
4               1.0000  0.045375    1.000000  1.000000  0.000000         inf
..                 ...       ...         ...       ...       ...         ...
76              0.1615  0.011875    0.124509  0.770951 -0.003528    0.957748
77              0.1615  0.011875    0.124509  0.770951 -0.003528    0.957748
78              1.0000  0.012625    1.000000  1.000000  0.000000         inf
79              0.1615  0.012625    0.146589  0.907673 -0.001284    0.982528
80              0.1615  0.012625    0.146589  0.907673 -0.001284    0.982528

    zhangs_metric
0        0.000000
1        0.000000
2        0.000000
3        0.000000
4        0.000000
..            ...
76      -0.247228
77      -0.247228
78       0.000000
79      -0.100156
80      -0.100156

[81 rows x 10 columns]
```

In [249...
```
#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.

msv_values = [0.001, 0.005, 0.01, 0.05]
mct_values = [0.05, 0.075, 0.1]
heatmap = pd.DataFrame(index=mct_values, columns=msv_values)
for msv in msv_values:
    frequent_itemsets = apriori(df_trans_encoded, min_support=msv, use_colnames=True)
    for mct in mct_values:
        rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=mct)
        heatmap.at[mct, msv] = len(rules)
sns.heatmap(heatmap_data, annot=True, cmap='magma', fmt='g', linewidths=.5)
plt.title('Association Rules Count Heatmap')
plt.xlabel('Minimum Support Value (msv)')
plt.ylabel('Minimum Confidence Threshold (mct)')
plt.show()
```



In [272...
```
#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

rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.1)
print(rules[rules['confidence'] == rules['confidence'].max()])


#What is that confidence value?
print(rules['confidence'].max())
#1.0
```

```
                         antecedents consequents  antecedent support  \
0                         (UHT-milk)   (missing)            0.021750
1                     (baking powder)   (missing)            0.008250
2                             (beef)   (missing)            0.032750
3                          (berries)   (missing)            0.021250
4                        (beverages)   (missing)            0.016250
..                               ...         ...                 ...
205                   (yogurt, soda)   (missing)            0.005750
206       (whole milk, tropical fruit)   (missing)            0.009375
209           (yogurt, tropical fruit)   (missing)            0.005500
210  (whipped/sour cream, whole milk)   (missing)            0.005750
213             (whole milk, yogurt)   (missing)            0.012625

     consequent support   support  confidence  lift  leverage  conviction  \
0                   1.0  0.021750         1.0   1.0       0.0         inf
1                   1.0  0.008250         1.0   1.0       0.0         inf
2                   1.0  0.032750         1.0   1.0       0.0         inf
3                   1.0  0.021250         1.0   1.0       0.0         inf
4                   1.0  0.016250         1.0   1.0       0.0         inf
..                  ...       ...         ...   ...       ...         ...
205                 1.0  0.005750         1.0   1.0       0.0         inf
206                 1.0  0.009375         1.0   1.0       0.0         inf
209                 1.0  0.005500         1.0   1.0       0.0         inf
210                 1.0  0.005750         1.0   1.0       0.0         inf
213                 1.0  0.012625         1.0   1.0       0.0         inf

     zhangs_metric
0              0.0
1              0.0
2              0.0
3              0.0
4              0.0
..             ...
205            0.0
206            0.0
209            0.0
210            0.0
213            0.0

[132 rows x 10 columns]
1.0
```

In [141...
```
#Construct a 4-class classification model using a convolutional neural network with the following simple architecture
import tensorflow as tf
import os
import imghdr
import pandas as pd
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
import numpy as np
import cv2
from pathlib import Path
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from tensorflow import keras


def load_and_preprocess_image(file_path):
    img = cv2.imread(file_path)

    img = img / 255.0
    return img


image_folder = r"C:\Users\puspa\OneDrive\Desktop\Cropped (1)\Cropped"
labels = ["n02092002-Scottish_deerhound", "n02093991-Irish_terrier", "n02097474-Tibetan_terrier", "n02106166-Border_collie"]

x_train_paths = [os.path.join(image_folder, f"{label}/{file}") for label in labels for file in os.listdir(os.path.join(image_folder, label))]
x_train = np.array([load_and_preprocess_image(file_path) for file_path in x_train_paths])
y_train_labels = [label for label in labels for _ in os.listdir(os.path.join(image_folder, label))]
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train_labels)
y_train = tf.keras.utils.to_categorical(y_train_encoded, len(labels))

# Build the CNN model
model = keras.Sequential()

# i. Convolutional Layer with 8 3 × 3 filters
model.add(layers.Conv2D(8, (3, 3), activation='relu', input_shape=(100, 100, 3)))

# ii.  max pooling with 2 × 2 pool size
model.add(layers.MaxPooling2D((2, 2)))

# iii. Flatten the Tensor
model.add(layers.Flatten())

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

# v. Output layer  using 'softmax' activation function
model.add(layers.Dense(len(labels), activation='softmax'))

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

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val))
```

```
Epoch 1/20
21/21 [==============================] - 1s 26ms/step - loss: 1.6652 - accuracy: 0.2773 - val_loss: 1.3279 - val_accuracy: 0.3230
Epoch 2/20
21/21 [==============================] - 0s 15ms/step - loss: 1.3007 - accuracy: 0.3583 - val_loss: 1.2848 - val_accuracy: 0.3975
Epoch 3/20
21/21 [==============================] - 0s 16ms/step - loss: 1.2850 - accuracy: 0.3692 - val_loss: 1.2583 - val_accuracy: 0.3602
Epoch 4/20
21/21 [==============================] - 0s 15ms/step - loss: 1.2442 - accuracy: 0.3925 - val_loss: 1.2468 - val_accuracy: 0.3789
Epoch 5/20
21/21 [==============================] - 0s 15ms/step - loss: 1.1905 - accuracy: 0.4252 - val_loss: 1.2038 - val_accuracy: 0.4099
Epoch 6/20
21/21 [==============================] - 0s 16ms/step - loss: 1.1557 - accuracy: 0.4393 - val_loss: 1.1935 - val_accuracy: 0.4224
Epoch 7/20
21/21 [==============================] - 0s 15ms/step - loss: 1.1246 - accuracy: 0.4377 - val_loss: 1.2318 - val_accuracy: 0.4099
Epoch 8/20
21/21 [==============================] - 0s 15ms/step - loss: 1.1326 - accuracy: 0.4330 - val_loss: 1.1620 - val_accuracy: 0.4161
Epoch 9/20
21/21 [==============================] - 0s 15ms/step - loss: 1.0800 - accuracy: 0.4813 - val_loss: 1.1457 - val_accuracy: 0.4224
Epoch 10/20
21/21 [==============================] - 0s 15ms/step - loss: 1.0449 - accuracy: 0.5093 - val_loss: 1.2774 - val_accuracy: 0.3478
Epoch 11/20
21/21 [==============================] - 0s 16ms/step - loss: 1.0665 - accuracy: 0.4844 - val_loss: 1.1424 - val_accuracy: 0.3975
Epoch 12/20
21/21 [==============================] - 0s 17ms/step - loss: 0.9978 - accuracy: 0.5000 - val_loss: 1.1235 - val_accuracy: 0.4286
Epoch 13/20
21/21 [==============================] - 0s 16ms/step - loss: 0.9690 - accuracy: 0.5498 - val_loss: 1.1376 - val_accuracy: 0.4410
Epoch 14/20
21/21 [==============================] - 0s 15ms/step - loss: 0.9414 - accuracy: 0.5405 - val_loss: 1.1112 - val_accuracy: 0.4286
Epoch 15/20
21/21 [==============================] - 0s 15ms/step - loss: 0.9481 - accuracy: 0.5358 - val_loss: 1.2253 - val_accuracy: 0.4472
Epoch 16/20
21/21 [==============================] - 0s 15ms/step - loss: 0.9348 - accuracy: 0.5701 - val_loss: 1.1011 - val_accuracy: 0.4534
Epoch 17/20
21/21 [==============================] - 0s 15ms/step - loss: 0.8717 - accuracy: 0.5857 - val_loss: 1.0792 - val_accuracy: 0.4410
Epoch 18/20
21/21 [==============================] - 0s 15ms/step - loss: 0.8496 - accuracy: 0.5717 - val_loss: 1.0816 - val_accuracy: 0.5714
Epoch 19/20
21/21 [==============================] - 0s 16ms/step - loss: 0.8292 - accuracy: 0.6075 - val_loss: 1.0609 - val_accuracy: 0.5901
Epoch 20/20
21/21 [==============================] - 0s 16ms/step - loss: 0.8082 - accuracy: 0.6075 - val_loss: 1.1597 - val_accuracy: 0.5217
```

In [143...
```
val_loss, val_acc = model.evaluate(X_val, y_val)

print("Validation Loss:", val_loss)
print("Validation Accuracy:", val_acc)

# Plot the learning curves
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Number of Epochs')
plt.ylabel('Training and Validation Accuracy')

plt.legend()
plt.show()
```

```
6/6 [==============================] - 0s 5ms/step - loss: 1.1597 - accuracy: 0.5217
Validation Loss: 1.1596601009368896
Validation Accuracy: 0.52173912525177
```

In [158...
```
#Build the model
input_shape = (100, 100, 3)
num_classes = 4

model = keras.Sequential(
    [
        layers.Input(shape=(100, 100, 3)),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential_54"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_58 (Conv2D)          (None, 98, 98, 32)        896

 max_pooling2d_58 (MaxPooli  (None, 49, 49, 32)        0
 ng2D)

 conv2d_59 (Conv2D)          (None, 47, 47, 64)        18496

 max_pooling2d_59 (MaxPooli  (None, 23, 23, 64)        0
 ng2D)

 flatten_52 (Flatten)        (None, 33856)             0

 dropout_6 (Dropout)         (None, 33856)             0

 dense_98 (Dense)            (None, 4)                 135428

=================================================================
Total params: 154820 (604.77 KB)
Trainable params: 154820 (604.77 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [166...
```
# Build the CNN model with a 5x5 filter size
model_5x5 = Sequential()
model_5x5.add(Conv2D(8, (5, 5), activation='relu', input_shape=(100, 100, 3)))
model_5x5.add(MaxPooling2D((2, 2)))
model_5x5.add(Flatten())
model_5x5.add(Dense(16, activation='relu'))
model_5x5.add(Dense(len(labels), activation='softmax'))


model_5x5.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
history_5x5 = model_5x5.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val))

# Build the CNN model with a 7x7 filter size
model_7x7 = Sequential()
model_7x7.add(Conv2D(8, (7, 7), activation='relu', input_shape=(100, 100, 3)))
model_7x7.add(MaxPooling2D((2, 2)))
model_7x7.add(Flatten())
model_7x7.add(Dense(16, activation='relu'))
model_7x7.add(Dense(len(labels), activation='softmax'))

model_7x7.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
history_7x7 = model_7x7.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val))

# Plot the learning curves for both models
plt.figure(figsize=(12, 4))

# Plot for the CNN using filter sizes: 5x5
plt.subplot(1, 2, 1)
plt.plot(range(1, 21), history_5x5.history['accuracy'], label='Training Accuracy (5x5)')
plt.plot(range(1, 21), history_5x5.history['val_accuracy'], label='Validation Accuracy (5x5)')
plt.xlabel('Number of Epochs')
plt.ylabel('Training and Validation accuracy')
plt.legend()

# Plot for the CNN using filter sizes: 7x7
plt.subplot(1, 2, 2)
plt.plot(range(1, 21), history_7x7.history['accuracy'], label='Training Accuracy (7x7)')
plt.plot(range(1, 21), history_7x7.history['val_accuracy'], label='Validation Accuracy (7x7)')
plt.xlabel('Number of Epochs')
plt.ylabel('Training and Validation accuracy')
plt.legend()

plt.tight_layout()
plt.show()

     #Talking about the first graph
# Increasing Training Accuracy means it is learning the training data well and may even be fitting the noise in the data.
# The gap between the training and validation accuracies in between indicates overfitting.
# Because it might be fitting the noise or specific details of the training set that do not generalise well.

     #Talking about the second and third graph
# Only Training accuracy is increasing that means this model is learning from the training data for both using 5*5 and 7*7 models.
#But if we talk about validation accuracy it has fluctuations and validation accuracy is significantly lower than training accuracy,has gap so it is overfitting.
 #The  second and third models seems to be fitting the training data closely but fails to generalize to unseen data, as reflected by the lower and fluctuating validation accuracy.
```

```
Epoch 1/20
21/21 [==============================] - 1s 33ms/step - loss: 1.5130 - accuracy: 0.2928 - val_loss: 1.3636 - val_accuracy: 0.3913
Epoch 2/20
21/21 [==============================] - 0s 20ms/step - loss: 1.2704 - accuracy: 0.4579 - val_loss: 1.2248 - val_accuracy: 0.4224
Epoch 3/20
21/21 [==============================] - 0s 20ms/step - loss: 1.0866 - accuracy: 0.5467 - val_loss: 1.1767 - val_accuracy: 0.4969
Epoch 4/20
21/21 [==============================] - 0s 19ms/step - loss: 0.9881 - accuracy: 0.6340 - val_loss: 1.1820 - val_accuracy: 0.5093
Epoch 5/20
21/21 [==============================] - 0s 19ms/step - loss: 0.8887 - accuracy: 0.6807 - val_loss: 1.1077 - val_accuracy: 0.5093
Epoch 6/20
21/21 [==============================] - 0s 19ms/step - loss: 0.7589 - accuracy: 0.7461 - val_loss: 1.1032 - val_accuracy: 0.5093
Epoch 7/20
21/21 [==============================] - 0s 19ms/step - loss: 0.7311 - accuracy: 0.7617 - val_loss: 1.1989 - val_accuracy: 0.5466
Epoch 8/20
21/21 [==============================] - 0s 19ms/step - loss: 0.5955 - accuracy: 0.8333 - val_loss: 1.1042 - val_accuracy: 0.6087
Epoch 9/20
21/21 [==============================] - 0s 19ms/step - loss: 0.4863 - accuracy: 0.8801 - val_loss: 1.0081 - val_accuracy: 0.5714
Epoch 10/20
21/21 [==============================] - 0s 20ms/step - loss: 0.4197 - accuracy: 0.8988 - val_loss: 1.1735 - val_accuracy: 0.6273
Epoch 11/20
21/21 [==============================] - 0s 19ms/step - loss: 0.3213 - accuracy: 0.9315 - val_loss: 1.1043 - val_accuracy: 0.5776
Epoch 12/20
21/21 [==============================] - 0s 20ms/step - loss: 0.2911 - accuracy: 0.9315 - val_loss: 1.0427 - val_accuracy: 0.5776
Epoch 13/20
21/21 [==============================] - 0s 21ms/step - loss: 0.2373 - accuracy: 0.9502 - val_loss: 1.1114 - val_accuracy: 0.5901
Epoch 14/20
21/21 [==============================] - 0s 21ms/step - loss: 0.2170 - accuracy: 0.9517 - val_loss: 1.0594 - val_accuracy: 0.5963
Epoch 15/20
21/21 [==============================] - 0s 19ms/step - loss: 0.1314 - accuracy: 0.9891 - val_loss: 1.1783 - val_accuracy: 0.5652
Epoch 16/20
21/21 [==============================] - 0s 18ms/step - loss: 0.1903 - accuracy: 0.9611 - val_loss: 1.0193 - val_accuracy: 0.5901
Epoch 17/20
21/21 [==============================] - 0s 19ms/step - loss: 0.0954 - accuracy: 0.9907 - val_loss: 1.1091 - val_accuracy: 0.5652
Epoch 18/20
21/21 [==============================] - 0s 20ms/step - loss: 0.1247 - accuracy: 0.9844 - val_loss: 1.1269 - val_accuracy: 0.5652
Epoch 19/20
21/21 [==============================] - 0s 19ms/step - loss: 0.0769 - accuracy: 0.9922 - val_loss: 1.1187 - val_accuracy: 0.5714
Epoch 20/20
21/21 [==============================] - 0s 19ms/step - loss: 0.0545 - accuracy: 0.9969 - val_loss: 1.1777 - val_accuracy: 0.5776
Epoch 1/20
21/21 [==============================] - 1s 35ms/step - loss: 1.4464 - accuracy: 0.3178 - val_loss: 1.2994 - val_accuracy: 0.3106
Epoch 2/20
21/21 [==============================] - 1s 27ms/step - loss: 1.2172 - accuracy: 0.3863 - val_loss: 1.1316 - val_accuracy: 0.4534
Epoch 3/20
21/21 [==============================] - 1s 29ms/step - loss: 0.9729 - accuracy: 0.5903 - val_loss: 0.9870 - val_accuracy: 0.5466
Epoch 4/20
21/21 [==============================] - 1s 26ms/step - loss: 0.7772 - accuracy: 0.6931 - val_loss: 0.9777 - val_accuracy: 0.5652
Epoch 5/20
21/21 [==============================] - 1s 25ms/step - loss: 0.6923 - accuracy: 0.7305 - val_loss: 0.9011 - val_accuracy: 0.6149
Epoch 6/20
21/21 [==============================] - 1s 25ms/step - loss: 0.5666 - accuracy: 0.8022 - val_loss: 1.2987 - val_accuracy: 0.5528
Epoch 7/20
21/21 [==============================] - 1s 32ms/step - loss: 0.5381 - accuracy: 0.8224 - val_loss: 0.8754 - val_accuracy: 0.6522
Epoch 8/20
21/21 [==============================] - 1s 25ms/step - loss: 0.3823 - accuracy: 0.8863 - val_loss: 1.0019 - val_accuracy: 0.5901
Epoch 9/20
21/21 [==============================] - 1s 27ms/step - loss: 0.3153 - accuracy: 0.8988 - val_loss: 0.8943 - val_accuracy: 0.6273
Epoch 10/20
21/21 [==============================] - 1s 26ms/step - loss: 0.2370 - accuracy: 0.9424 - val_loss: 0.9086 - val_accuracy: 0.6460
Epoch 11/20
21/21 [==============================] - 1s 38ms/step - loss: 0.1808 - accuracy: 0.9657 - val_loss: 0.9625 - val_accuracy: 0.6211
Epoch 12/20
21/21 [==============================] - 1s 27ms/step - loss: 0.1357 - accuracy: 0.9782 - val_loss: 1.1886 - val_accuracy: 0.5714
Epoch 13/20
21/21 [==============================] - 1s 27ms/step - loss: 0.1160 - accuracy: 0.9844 - val_loss: 1.0348 - val_accuracy: 0.6087
Epoch 14/20
21/21 [==============================] - 1s 24ms/step - loss: 0.0770 - accuracy: 0.9938 - val_loss: 1.1431 - val_accuracy: 0.6087
Epoch 15/20
21/21 [==============================] - 1s 24ms/step - loss: 0.0990 - accuracy: 0.9844 - val_loss: 1.0733 - val_accuracy: 0.6149
Epoch 16/20
21/21 [==============================] - 1s 24ms/step - loss: 0.0548 - accuracy: 0.9984 - val_loss: 1.2277 - val_accuracy: 0.6273
Epoch 17/20
21/21 [==============================] - 1s 24ms/step - loss: 0.0383 - accuracy: 1.0000 - val_loss: 1.1991 - val_accuracy: 0.6149
Epoch 18/20
21/21 [==============================] - 1s 24ms/step - loss: 0.0305 - accuracy: 1.0000 - val_loss: 1.2886 - val_accuracy: 0.6087
Epoch 19/20
21/21 [==============================] - 1s 25ms/step - loss: 0.0220 - accuracy: 1.0000 - val_loss: 1.2742 - val_accuracy: 0.6335
Epoch 20/20
21/21 [==============================] - 1s 25ms/step - loss: 0.0181 - accuracy: 1.0000 - val_loss: 1.3058 - val_accuracy: 0.6149
```


In [ ]: