Homework Assignment 3

In [96]:
```
grocery_data_path= r'C:\Users\injam\Downloads\Grocery_Items_25.csv'
```

In [97]:
```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

data= pd.read_csv(grocery_data_path)

# Drop any columns with NaN values
g_df = [row.dropna().tolist() for index, row in data.iterrows()]

# Convert the DataFrame into a transaction format using TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(grocery_df).transform(grocery_df)

df = pd.DataFrame(te_ary, columns=te.columns_)
df.head()
```

Out[97]:

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	bags	baking powder	bathroom cleaner	beef	berries	beverages	...	turkey	vineg
0	False	False	False	False	False	True	False	False	False	False	...	False	Fa
1	False	False	False	False	False	False	False	False	False	False	...	False	Fa
2	False	False	False	False	False	False	False	False	False	False	...	False	Fa
3	False	False	False	False	False	True	False	False	False	False	...	False	Fa
4	False	False	False	False	False	False	False	False	False	False	...	False	Fa

5 rows × 165 columns

Using minimum support = 0.01 and minimum confidence threshold = 0.1, what are the association rules you can extract from your dataset?

In [98]:
```
from mlxtend.frequent_patterns import apriori,association_rules

items = apriori(df, min_support=0.01, use_colnames=True)
association_rules(items, metric="confidence", min_threshold=0.1)
```

Out[98]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(soda)	(other vegetables)	0.103125	0.122000	0.010625	0.103030	0.844511	-0.001956	0.978851
1	(other vegetables)	(whole milk)	0.122000	0.154625	0.014250	0.116803	0.755397	-0.004614	0.957176
2	(rolls/buns)	(whole milk)	0.109625	0.154625	0.012750	0.116306	0.752178	-0.004201	0.956637
3	(soda)	(whole milk)	0.103125	0.154625	0.011875	0.115152	0.744715	-0.004071	0.955390
4	(yogurt)	(whole milk)	0.088125	0.154625	0.011375	0.129078	0.834781	-0.002251	0.970667

Use minimum support values (msv): 0.001, 0.005, 0.01 and minimum confidence threshold (mct): 0.05, 0.075, 0.1. For each pair (msv, mct), find the number of association rules extracted from the dataset. Construct a heatmap using Seaborn data visualization library (https://seaborn.

pydata.org/generated/seaborn.heatmap.html) to show the count results such that the x axis is msv and the y-axis is mct.

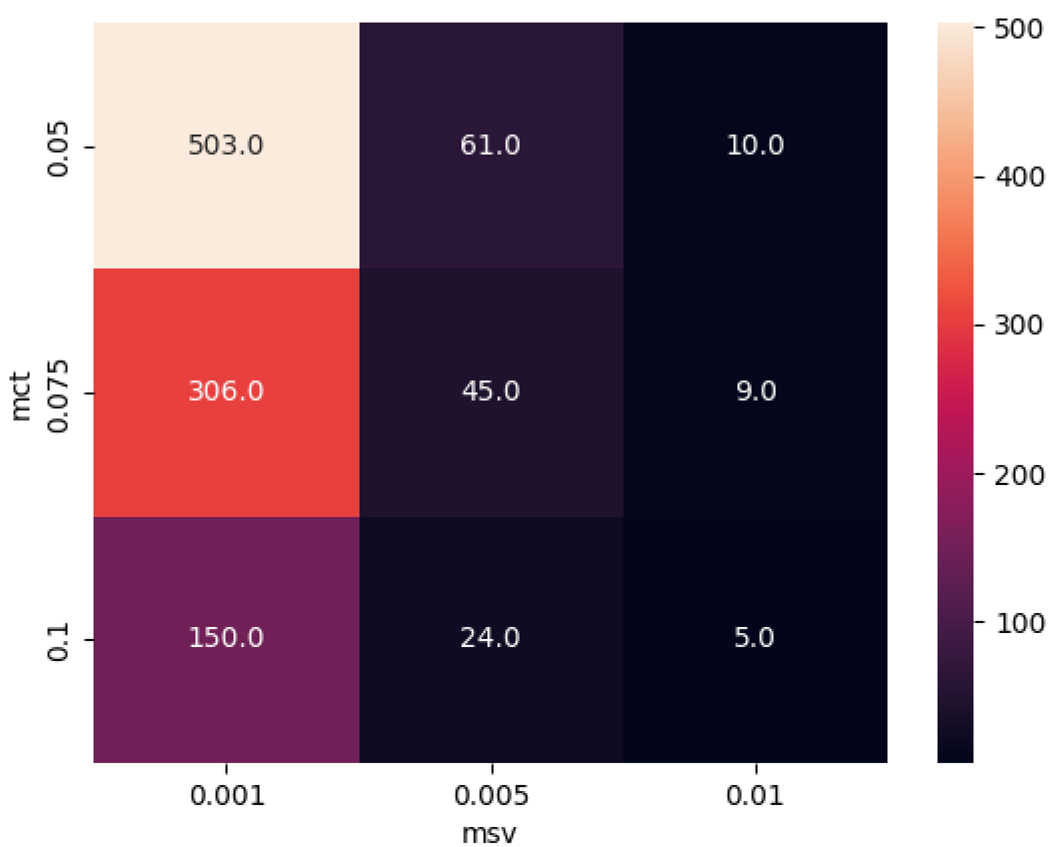
In [108…

```
import seaborn as sns
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

minimum_support_values = [0.001, 0.005, 0.01]
minimum_confidence_threshold = [0.05, 0.075, 0.1]

heatmap_data = pd.DataFrame({
    'msv': [i for i in minimum_support_values for _ in minimum_confidence_threshold],
    'mct': minimum_confidence_threshold * len(minimum_support_values),
    'count': [len(association_rules(apriori(df, min_support=i, use_colnames=True), metri
              for i in minimum_support_values for j in minimum_confidence_threshold]
})
print(heatmap_data)
heatmap_data = heatmap_data.pivot(index="mct", columns="msv", values="count")
sns.heatmap(heatmap_data, annot=True, fmt=".1f")
```

```
     msv    mct  count
0  0.001  0.050    503
1  0.001  0.075    306
2  0.001  0.100    150
3  0.005  0.050     61
4  0.005  0.075     45
5  0.005  0.100     24
6  0.010  0.050     10
7  0.010  0.075      9
8  0.010  0.100      5
```

Out[108]:

```
<Axes: xlabel='msv', ylabel='mct'>
```



Split the dataset into 50:50 (i.e., 2 equal subsets) and extract association rules for each data subset for minimum support = 0.005 and minimum confident threshold = 0.075. Show the association rules for both sets. Which association rules appeared in both sets (note that there could be none)?

In [99]:
```
set1 = df.iloc[:len(df)//2]
set2 = df.iloc[len(df)//2:]

items = apriori(set1, min_support=0.005, use_colnames=True)
r_1 = association_rules(items, metric="confidence", min_threshold=0.075)

items = apriori(set2, min_support=0.005, use_colnames=True)
r_2 = association_rules(items, metric="confidence", min_threshold=0.075)
common_rules = pd.merge(rules1, rules2,on=['antecedents', 'consequents'])
```

In [100…
```
r_1
```

Out[100]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(bottled beer)	(other vegetables)	0.04600	0.12175	0.00550	0.119565	0.982055	-0.000100	0.997519
1	(bottled beer)	(rolls/buns)	0.04600	0.11325	0.00575	0.125000	1.103753	0.000541	1.013429
2	(bottled beer)	(whole milk)	0.04600	0.15150	0.00750	0.163043	1.076195	0.000531	1.013792
3	(bottled water)	(rolls/buns)	0.06075	0.11325	0.00575	0.094650	0.835763	-0.001130	0.979456
4	(bottled water)	(soda)	0.06075	0.10950	0.00550	0.090535	0.826803	-0.001152	0.979147
5	(bottled water)	(whole milk)	0.06075	0.15150	0.00575	0.094650	0.624754	-0.003454	0.937207
6	(citrus fruit)	(whole milk)	0.05625	0.15150	0.00750	0.133333	0.880088	-0.001022	0.979038
7	(citrus fruit)	(yogurt)	0.05625	0.09000	0.00575	0.102222	1.135802	0.000687	1.013614
8	(frankfurter)	(other vegetables)	0.03600	0.12175	0.00525	0.145833	1.197810	0.000867	1.028195
9	(frankfurter)	(whole milk)	0.03600	0.15150	0.00525	0.145833	0.962596	-0.000204	0.993366
10	(newspapers)	(other vegetables)	0.04125	0.12175	0.00550	0.133333	1.095140	0.000478	1.013365
11	(newspapers)	(whole milk)	0.04125	0.15150	0.00675	0.163636	1.080108	0.000501	1.014511
12	(pip fruit)	(other vegetables)	0.04550	0.12175	0.00600	0.131868	1.083106	0.000460	1.011655
13	(rolls/buns)	(other vegetables)	0.11325	0.12175	0.00975	0.086093	0.707127	-0.004038	0.960984
14	(other vegetables)	(rolls/buns)	0.12175	0.11325	0.00975	0.080082	0.707127	-0.004038	0.963945
15	(root vegetables)	(other vegetables)	0.07200	0.12175	0.00650	0.090278	0.741501	-0.002266	0.965405
16	(sausage)	(other vegetables)	0.06000	0.12175	0.00500	0.083333	0.684463	-0.002305	0.958091
17	(soda)	(other vegetables)	0.10950	0.12175	0.01100	0.100457	0.825106	-0.002332	0.976329
18	(other vegetables)	(soda)	0.12175	0.10950	0.01100	0.090349	0.825106	-0.002332	0.978947
19	(tropical fruit)	(other vegetables)	0.07225	0.12175	0.00675	0.093426	0.767356	-0.002046	0.968757
20	(whole milk)	(other vegetables)	0.15150	0.12175	0.01525	0.100660	0.826777	-0.003195	0.976550
21	(other vegetables)	(whole milk)	0.12175	0.15150	0.01525	0.125257	0.826777	-0.003195	0.969999

22	(yogurt)	(other vegetables)	0.09000	0.12175	0.00900	0.100000	0.821355	-0.001957	0.975833
23	(pastry)	(rolls/buns)	0.05000	0.11325	0.00500	0.100000	0.883002	-0.000663	0.985278
24	(pastry)	(soda)	0.05000	0.10950	0.00500	0.100000	0.913242	-0.000475	0.989444
25	(pastry)	(whole milk)	0.05000	0.15150	0.00600	0.120000	0.792079	-0.001575	0.964205
26	(pip fruit)	(rolls/buns)	0.04550	0.11325	0.00525	0.115385	1.018849	0.000097	1.002413
27	(pip fruit)	(whole milk)	0.04550	0.15150	0.00600	0.131868	0.870417	-0.000893	0.977386
28	(root vegetables)	(rolls/buns)	0.07200	0.11325	0.00625	0.086806	0.766495	-0.001904	0.971042
29	(sausage)	(rolls/buns)	0.06000	0.11325	0.00500	0.083333	0.735835	-0.001795	0.967364
30	(shopping bags)	(rolls/buns)	0.05100	0.11325	0.00575	0.112745	0.995542	-0.000026	0.999431
31	(soda)	(rolls/buns)	0.10950	0.11325	0.00950	0.086758	0.766075	-0.002901	0.970991
32	(rolls/buns)	(soda)	0.11325	0.10950	0.00950	0.083885	0.766075	-0.002901	0.972040
33	(tropical fruit)	(rolls/buns)	0.07225	0.11325	0.00750	0.103806	0.916611	-0.000682	0.989462
34	(whole milk)	(rolls/buns)	0.15150	0.11325	0.01350	0.089109	0.786834	-0.003657	0.973497
35	(rolls/buns)	(whole milk)	0.11325	0.15150	0.01350	0.119205	0.786834	-0.003657	0.963335
36	(yogurt)	(rolls/buns)	0.09000	0.11325	0.00850	0.094444	0.833947	-0.001692	0.979233
37	(rolls/buns)	(yogurt)	0.11325	0.09000	0.00850	0.075055	0.833947	-0.001692	0.983842
38	(shopping bags)	(root vegetables)	0.05100	0.07200	0.00500	0.098039	1.361656	0.001328	1.028870
39	(root vegetables)	(soda)	0.07200	0.10950	0.00550	0.076389	0.697615	-0.002384	0.964150
40	(root vegetables)	(whole milk)	0.07200	0.15150	0.00675	0.093750	0.618812	-0.004158	0.936276
41	(sausage)	(soda)	0.06000	0.10950	0.00650	0.108333	0.989346	-0.000070	0.998692
42	(sausage)	(whole milk)	0.06000	0.15150	0.00850	0.141667	0.935094	-0.000590	0.988544
43	(sausage)	(yogurt)	0.06000	0.09000	0.00550	0.091667	1.018519	0.000100	1.001835
44	(shopping bags)	(soda)	0.05100	0.10950	0.00600	0.117647	1.074402	0.000416	1.009233
45	(shopping bags)	(whole milk)	0.05100	0.15150	0.00550	0.107843	0.711836	-0.002226	0.951066
46	(tropical fruit)	(soda)	0.07225	0.10950	0.00600	0.083045	0.758402	-0.001911	0.971149
47	(soda)	(whole milk)	0.10950	0.15150	0.01100	0.100457	0.663080	-0.005589	0.943256
48	(yogurt)	(soda)	0.09000	0.10950	0.00675	0.075000	0.684932	-0.003105	0.962703
49	(tropical fruit)	(whole milk)	0.07225	0.15150	0.00975	0.134948	0.890747	-0.001196	0.980866
50	(yogurt)	(tropical fruit)	0.09000	0.07225	0.00700	0.077778	1.076509	0.000498	1.005994
51	(tropical fruit)	(yogurt)	0.07225	0.09000	0.00700	0.096886	1.076509	0.000498	1.007625
52	(whipped/sour cream)	(whole milk)	0.04350	0.15150	0.00500	0.114943	0.758697	-0.001590	0.958695
53	(yogurt)	(whole milk)	0.09000	0.15150	0.01175	0.130556	0.861753	-0.001885	0.975911
54	(whole milk)	(yogurt)	0.15150	0.09000	0.01175	0.077558	0.861753	-0.001885	0.986512

In [101…

```
r_2
```

	antecedents	consequents	antecedent	consequent	support	confidence	lift	leverage	conviction

Out[101]:

			support	support					
0	(beef)	(whole milk)	0.03475	0.15775	0.00525	0.151079	0.957712	-0.000232	0.992142
1	(bottled beer)	(other vegetables)	0.04400	0.12225	0.00500	0.113636	0.929541	-0.000379	0.990282
2	(bottled beer)	(whole milk)	0.04400	0.15775	0.00550	0.125000	0.792393	-0.001441	0.962571
3	(bottled water)	(other vegetables)	0.06050	0.12225	0.00625	0.103306	0.845037	-0.001146	0.978873
4	(bottled water)	(soda)	0.06050	0.09675	0.00525	0.086777	0.896918	-0.000603	0.989079
5	(bottled water)	(whole milk)	0.06050	0.15775	0.00825	0.136364	0.864429	-0.001294	0.975237
6	(butter)	(whole milk)	0.03500	0.15775	0.00525	0.150000	0.950872	-0.000271	0.990882
7	(canned beer)	(whole milk)	0.05075	0.15775	0.00625	0.123153	0.780683	-0.001756	0.960544
8	(canned beer)	(yogurt)	0.05075	0.08625	0.00500	0.098522	1.142286	0.000623	1.013613
9	(citrus fruit)	(other vegetables)	0.05325	0.12225	0.00525	0.098592	0.806475	-0.001260	0.973754
10	(citrus fruit)	(whole milk)	0.05325	0.15775	0.00675	0.126761	0.803553	-0.001650	0.964512
11	(citrus fruit)	(yogurt)	0.05325	0.08625	0.00525	0.098592	1.143090	0.000657	1.013691
12	(domestic eggs)	(whole milk)	0.03475	0.15775	0.00725	0.208633	1.322555	0.001768	1.064298
13	(frankfurter)	(other vegetables)	0.03900	0.12225	0.00550	0.141026	1.153584	0.000732	1.021858
14	(newspapers)	(whole milk)	0.03725	0.15775	0.00550	0.147651	0.935981	-0.000376	0.988152
15	(pip fruit)	(other vegetables)	0.04675	0.12225	0.00500	0.106952	0.874862	-0.000715	0.982870
16	(rolls/buns)	(other vegetables)	0.10600	0.12225	0.01000	0.094340	0.771694	-0.002958	0.969182
17	(other vegetables)	(rolls/buns)	0.12225	0.10600	0.01000	0.081800	0.771694	-0.002958	0.973644
18	(sausage)	(other vegetables)	0.06250	0.12225	0.00600	0.096000	0.785276	-0.001641	0.970962
19	(shopping bags)	(other vegetables)	0.04650	0.12225	0.00575	0.123656	1.011500	0.000065	1.001604
20	(soda)	(other vegetables)	0.09675	0.12225	0.01025	0.105943	0.866611	-0.001578	0.981761
21	(other vegetables)	(soda)	0.12225	0.09675	0.01025	0.083845	0.866611	-0.001578	0.985914
22	(tropical fruit)	(other vegetables)	0.05900	0.12225	0.00525	0.088983	0.727878	-0.001963	0.963484
23	(whipped/sour cream)	(other vegetables)	0.04500	0.12225	0.00500	0.111111	0.908884	-0.000501	0.987469
24	(whole milk)	(other vegetables)	0.15775	0.12225	0.01325	0.083994	0.687065	-0.006035	0.958236
25	(other vegetables)	(whole milk)	0.12225	0.15775	0.01325	0.108384	0.687065	-0.006035	0.944634
26	(yogurt)	(other vegetables)	0.08625	0.12225	0.00850	0.098551	0.806141	-0.002044	0.973710
27	(pastry)	(whole milk)	0.05325	0.15775	0.00600	0.112676	0.714270	-0.002400	0.949202
28	(pip fruit)	(whole milk)	0.04675	0.15775	0.00625	0.133690	0.847479	-0.001125	0.972227

29	(root vegetables)	(rolls/buns)	0.06950	0.10600	0.00575	0.082734	0.780508	-0.001617	0.974635
30	(sausage)	(rolls/buns)	0.06250	0.10600	0.00600	0.096000	0.905660	-0.000625	0.988938
31	(soda)	(rolls/buns)	0.09675	0.10600	0.00800	0.082687	0.780069	-0.002256	0.974586
32	(rolls/buns)	(soda)	0.10600	0.09675	0.00800	0.075472	0.780069	-0.002256	0.976985
33	(whole milk)	(rolls/buns)	0.15775	0.10600	0.01200	0.076070	0.717639	-0.004722	0.967605
34	(rolls/buns)	(whole milk)	0.10600	0.15775	0.01200	0.113208	0.717639	-0.004722	0.949771
35	(yogurt)	(rolls/buns)	0.08625	0.10600	0.00725	0.084058	0.793000	-0.001892	0.976044
36	(root vegetables)	(soda)	0.06950	0.09675	0.00575	0.082734	0.855130	-0.000974	0.984720
37	(root vegetables)	(tropical fruit)	0.06950	0.05900	0.00525	0.075540	1.280332	0.001150	1.017891
38	(tropical fruit)	(root vegetables)	0.05900	0.06950	0.00525	0.088983	1.280332	0.001150	1.021386
39	(root vegetables)	(whole milk)	0.06950	0.15775	0.00850	0.122302	0.775291	-0.002464	0.959613
40	(sausage)	(whole milk)	0.06250	0.15775	0.00850	0.136000	0.862124	-0.001359	0.974826
41	(yogurt)	(sausage)	0.08625	0.06250	0.00675	0.078261	1.252174	0.001359	1.017099
42	(sausage)	(yogurt)	0.06250	0.08625	0.00675	0.108000	1.252174	0.001359	1.024383
43	(shopping bags)	(whole milk)	0.04650	0.15775	0.00600	0.129032	0.817954	-0.001335	0.967028
44	(soda)	(whole milk)	0.09675	0.15775	0.01275	0.131783	0.835391	-0.002512	0.970092
45	(whole milk)	(soda)	0.15775	0.09675	0.01275	0.080824	0.835391	-0.002512	0.982674
46	(tropical fruit)	(whole milk)	0.05900	0.15775	0.00550	0.093220	0.590937	-0.003807	0.928836
47	(yogurt)	(whole milk)	0.08625	0.15775	0.01100	0.127536	0.808471	-0.002606	0.965370

In [102…

```
common_rules
```

Out[102]:

	antecedents	consequents	antecedent support_x	consequent support_x	support_x	confidence_x	lift_x	leverage_x	conv
0	(bottled beer)	(other vegetables)	0.04600	0.12175	0.00550	0.119565	0.982055	-0.000100	0
1	(bottled beer)	(whole milk)	0.04600	0.15150	0.00750	0.163043	1.076195	0.000531	1
2	(bottled water)	(soda)	0.06075	0.10950	0.00550	0.090535	0.826803	-0.001152	0
3	(bottled water)	(whole milk)	0.06075	0.15150	0.00575	0.094650	0.624754	-0.003454	0
4	(citrus fruit)	(whole milk)	0.05625	0.15150	0.00750	0.133333	0.880088	-0.001022	0
5	(citrus fruit)	(yogurt)	0.05625	0.09000	0.00575	0.102222	1.135802	0.000687	1
6	(frankfurter)	(other vegetables)	0.03600	0.12175	0.00525	0.145833	1.197810	0.000867	1
7	(newspapers)	(whole milk)	0.04125	0.15150	0.00675	0.163636	1.080108	0.000501	1
8	(pip fruit)	(other vegetables)	0.04550	0.12175	0.00600	0.131868	1.083106	0.000460	1
9	(rolls/buns)	(other vegetables)	0.11325	0.12175	0.00975	0.086093	0.707127	-0.004038	0
10	(other vegetables)	(rolls/buns)	0.12175	0.11325	0.00975	0.080082	0.707127	-0.004038	0

11	(sausage)	(other vegetables)	0.06000	0.12175	0.00500	0.083333	0.684463	-0.002305	0
12	(soda)	(other vegetables)	0.10950	0.12175	0.01100	0.100457	0.825106	-0.002332	0
13	(other vegetables)	(soda)	0.12175	0.10950	0.01100	0.090349	0.825106	-0.002332	0
14	(tropical fruit)	(other vegetables)	0.07225	0.12175	0.00675	0.093426	0.767356	-0.002046	0
15	(whole milk)	(other vegetables)	0.15150	0.12175	0.01525	0.100660	0.826777	-0.003195	0
16	(other vegetables)	(whole milk)	0.12175	0.15150	0.01525	0.125257	0.826777	-0.003195	0
17	(yogurt)	(other vegetables)	0.09000	0.12175	0.00900	0.100000	0.821355	-0.001957	0
18	(pastry)	(whole milk)	0.05000	0.15150	0.00600	0.120000	0.792079	-0.001575	0
19	(pip fruit)	(whole milk)	0.04550	0.15150	0.00600	0.131868	0.870417	-0.000893	0
20	(root vegetables)	(rolls/buns)	0.07200	0.11325	0.00625	0.086806	0.766495	-0.001904	0
21	(sausage)	(rolls/buns)	0.06000	0.11325	0.00500	0.083333	0.735835	-0.001795	0
22	(soda)	(rolls/buns)	0.10950	0.11325	0.00950	0.086758	0.766075	-0.002901	0
23	(rolls/buns)	(soda)	0.11325	0.10950	0.00950	0.083885	0.766075	-0.002901	0
24	(whole milk)	(rolls/buns)	0.15150	0.11325	0.01350	0.089109	0.786834	-0.003657	0
25	(rolls/buns)	(whole milk)	0.11325	0.15150	0.01350	0.119205	0.786834	-0.003657	0
26	(yogurt)	(rolls/buns)	0.09000	0.11325	0.00850	0.094444	0.833947	-0.001692	0
27	(root vegetables)	(soda)	0.07200	0.10950	0.00550	0.076389	0.697615	-0.002384	0
28	(root vegetables)	(whole milk)	0.07200	0.15150	0.00675	0.093750	0.618812	-0.004158	0
29	(sausage)	(whole milk)	0.06000	0.15150	0.00850	0.141667	0.935094	-0.000590	0
30	(sausage)	(yogurt)	0.06000	0.09000	0.00550	0.091667	1.018519	0.000100	1
31	(shopping bags)	(whole milk)	0.05100	0.15150	0.00550	0.107843	0.711836	-0.002226	0
32	(soda)	(whole milk)	0.10950	0.15150	0.01100	0.100457	0.663080	-0.005589	0
33	(tropical fruit)	(whole milk)	0.07225	0.15150	0.00975	0.134948	0.890747	-0.001196	0
34	(yogurt)	(whole milk)	0.09000	0.15150	0.01175	0.130556	0.861753	-0.001885	0

ImageClassification using CNN

In [109…

```
directory = r'C:\Users\injam\Desktop\DM_Assignment_1\Cropped'
```

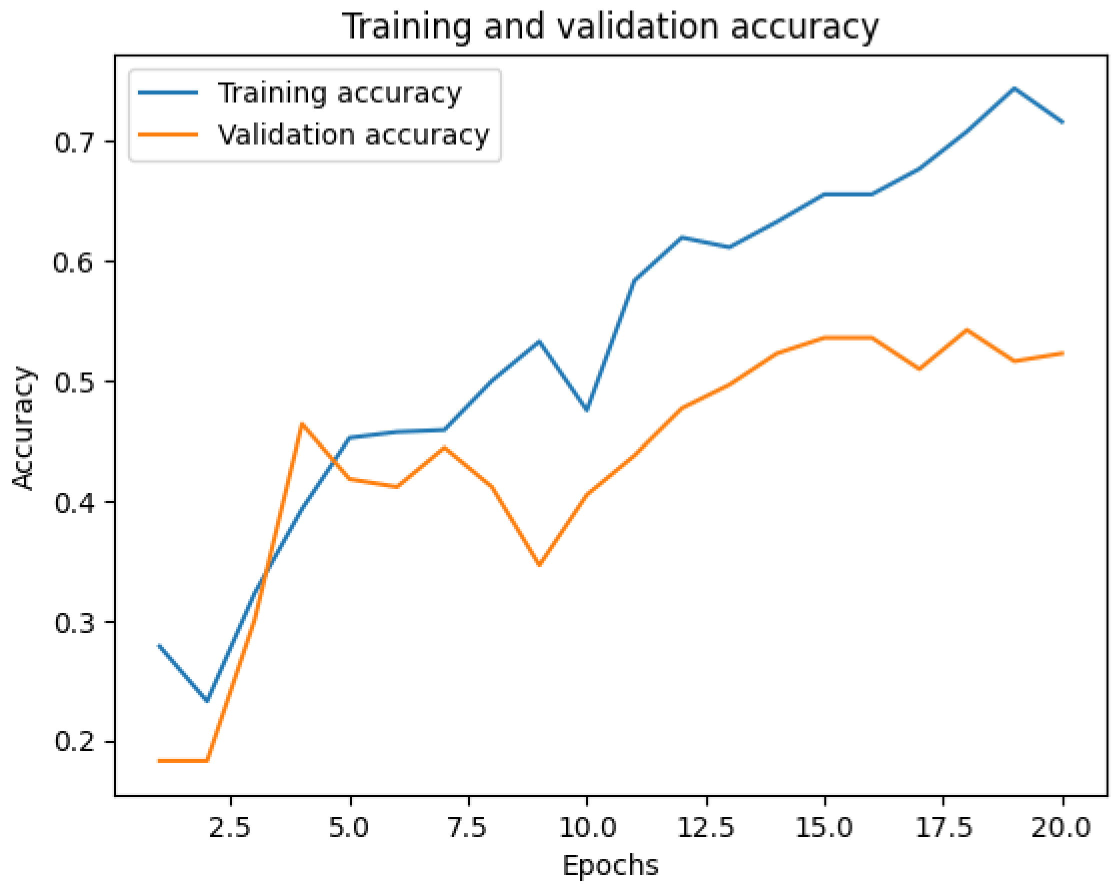
In [110…

```
import os
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
```

```
import warnings
warnings.filterwarnings("ignore")

directory1 = r'C:\Users\injam\Desktop\DM_Assignment_1\Cropped\n02093647-Bedlington_terri
directory2 = r'C:\Users\injam\Desktop\DM_Assignment_1\Cropped\n02099849-Chesapeake_Bay_r
directory3 = r'C:\Users\injam\Desktop\DM_Assignment_1\Cropped\n02100735-English_setter'
directory4 = r'C:\Users\injam\Desktop\DM_Assignment_1\Cropped\n02116738-African_hunting_

image_height, image_width = 128, 128

def plot_training_curves(history):
    train_accuracy = history.history['accuracy']
    val_accuracy = history.history['val_accuracy']
    epochs = range(1, len(train_accuracy) + 1)
    plt.plot(epochs, train_accuracy, label='Training accuracy')
    plt.plot(epochs, val_accuracy, label='Validation accuracy')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

def load_images_and_labels(folder):
    images = []
    labels = []
    for filename in os.listdir(folder):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            img = load_img(os.path.join(folder, filename), target_size=(image_height, im
            img_array = img_to_array(img)
            images.append(img_array)
            if folder == directory1:
                labels.append(0)
            elif folder == directory2:
                labels.append(1)
            elif folder == directory3:
                labels.append(2)
            elif folder == directory4:
                labels.append(3)
    return images, labels

class1_images, class1_labels = load_images_and_labels(directory1)
class2_images, class2_labels = load_images_and_labels(directory2)
class3_images, class3_labels = load_images_and_labels(directory3)
class4_images, class4_labels = load_images_and_labels(directory4)

images = np.concatenate([class1_images, class2_images, class3_images, class4_images], ax
labels = np.concatenate([class1_labels, class2_labels, class3_labels, class4_labels], ax

labels = to_categorical(labels)

X_train, X_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, random_

X_train = X_train / 255.0
X_val = X_val / 255.0

model = Sequential([
    Conv2D(8, (3, 3), activation='relu', input_shape=(image_height, image_width, 3)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(16, activation='relu'),
    Dense(4, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
```

```
               metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))
plot_training_curves(history)
```

```
Epoch 1/20
20/20 [==============================] - 5s 131ms/step - loss: 1.9130 - accuracy: 0.2787
- val_loss: 1.3715 - val_accuracy: 0.1830
Epoch 2/20
20/20 [==============================] - 2s 103ms/step - loss: 1.3300 - accuracy: 0.2328
- val_loss: 1.3285 - val_accuracy: 0.1830
Epoch 3/20
20/20 [==============================] - 2s 101ms/step - loss: 1.2720 - accuracy: 0.3230
- val_loss: 1.3705 - val_accuracy: 0.3007
Epoch 4/20
20/20 [==============================] - 2s 100ms/step - loss: 1.2553 - accuracy: 0.3934
- val_loss: 1.2988 - val_accuracy: 0.4641
Epoch 5/20
20/20 [==============================] - 2s 102ms/step - loss: 1.2144 - accuracy: 0.4525
- val_loss: 1.2759 - val_accuracy: 0.4183
Epoch 6/20
20/20 [==============================] - 2s 100ms/step - loss: 1.1868 - accuracy: 0.4574
- val_loss: 1.2758 - val_accuracy: 0.4118
Epoch 7/20
20/20 [==============================] - 2s 104ms/step - loss: 1.1726 - accuracy: 0.4590
- val_loss: 1.2421 - val_accuracy: 0.4444
Epoch 8/20
20/20 [==============================] - 2s 101ms/step - loss: 1.0601 - accuracy: 0.5000
- val_loss: 1.2370 - val_accuracy: 0.4118
Epoch 9/20
20/20 [==============================] - 2s 101ms/step - loss: 0.9757 - accuracy: 0.5328
- val_loss: 1.5956 - val_accuracy: 0.3464
Epoch 10/20
20/20 [==============================] - 2s 104ms/step - loss: 1.1354 - accuracy: 0.4754
- val_loss: 1.1959 - val_accuracy: 0.4052
Epoch 11/20
20/20 [==============================] - 2s 101ms/step - loss: 0.9258 - accuracy: 0.5836
- val_loss: 1.1351 - val_accuracy: 0.4379
Epoch 12/20
20/20 [==============================] - 2s 118ms/step - loss: 0.8599 - accuracy: 0.6197
- val_loss: 1.0653 - val_accuracy: 0.4771
Epoch 13/20
20/20 [==============================] - 3s 124ms/step - loss: 0.8351 - accuracy: 0.6115
- val_loss: 1.0714 - val_accuracy: 0.4967
Epoch 14/20
20/20 [==============================] - 3s 124ms/step - loss: 0.8160 - accuracy: 0.6328
- val_loss: 1.0933 - val_accuracy: 0.5229
Epoch 15/20
20/20 [==============================] - 3s 128ms/step - loss: 0.7960 - accuracy: 0.6557
- val_loss: 1.0567 - val_accuracy: 0.5359
Epoch 16/20
20/20 [==============================] - 2s 121ms/step - loss: 0.7799 - accuracy: 0.6557
- val_loss: 1.0595 - val_accuracy: 0.5359
Epoch 17/20
20/20 [==============================] - 2s 110ms/step - loss: 0.7445 - accuracy: 0.6770
- val_loss: 1.0973 - val_accuracy: 0.5098
Epoch 18/20
20/20 [==============================] - 2s 108ms/step - loss: 0.7187 - accuracy: 0.7082
- val_loss: 1.0720 - val_accuracy: 0.5425
Epoch 19/20
20/20 [==============================] - 2s 107ms/step - loss: 0.6884 - accuracy: 0.7443
- val_loss: 1.0884 - val_accuracy: 0.5163
Epoch 20/20
20/20 [==============================] - 2s 106ms/step - loss: 0.6745 - accuracy: 0.7164
- val_loss: 1.0892 - val_accuracy: 0.5229
```



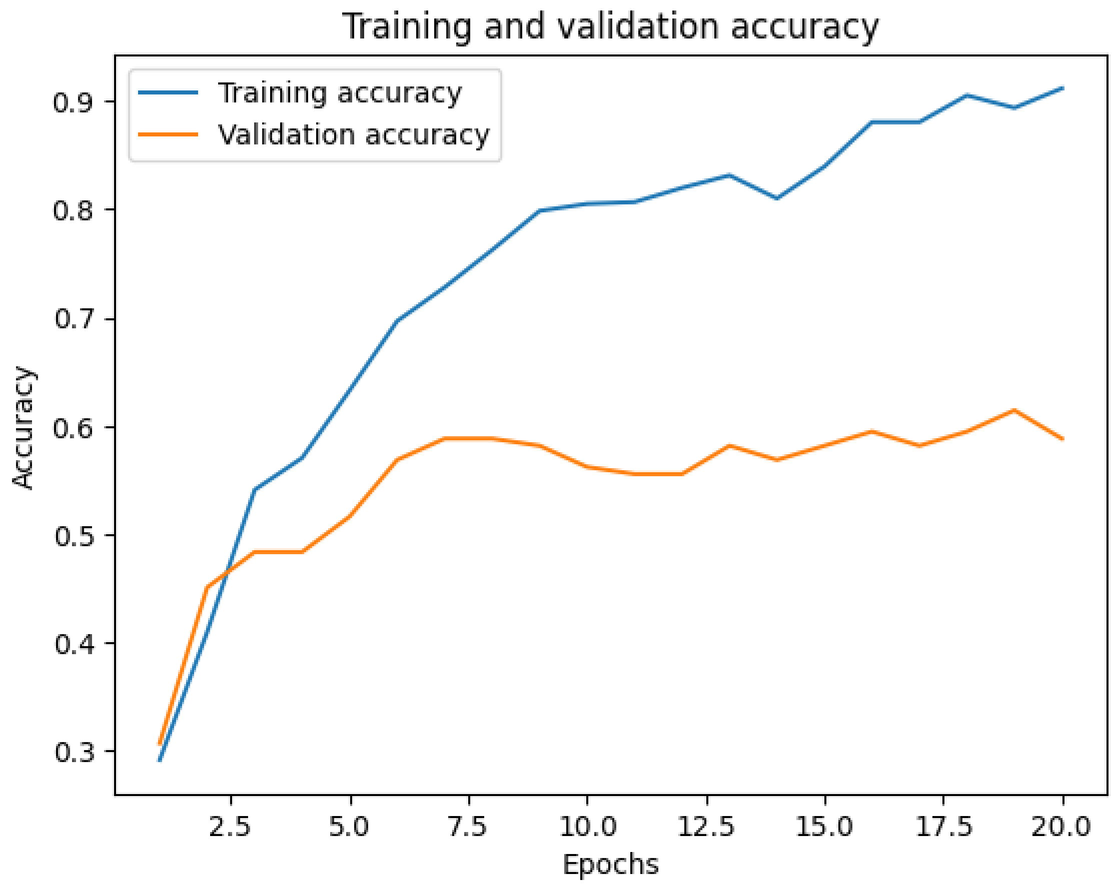
Train the CNN using 2 other number of nodes in the hidden layer (iv): 8 and 32 with all other parameters unchanged

In [112…

```
updated_model = Sequential([
    Conv2D(8, (3, 3), activation='relu', input_shape=(image_height, image_width, 3)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(8, activation='relu'),
    Dense(4, activation='softmax')
])

updated_model.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

training_history = updated_model.fit(X_train, y_train, epochs=20, validation_data=(X_val
plot_training_curves(training_history)
```

```
Epoch 1/20
20/20 [==============================] - 7s 129ms/step - loss: 1.4112 - accuracy: 0.2918
- val_loss: 1.2944 - val_accuracy: 0.3072
Epoch 2/20
20/20 [==============================] - 2s 102ms/step - loss: 1.2222 - accuracy: 0.4098
- val_loss: 1.1892 - val_accuracy: 0.4510
Epoch 3/20
20/20 [==============================] - 2s 104ms/step - loss: 1.0784 - accuracy: 0.5410
- val_loss: 1.1812 - val_accuracy: 0.4837
Epoch 4/20
20/20 [==============================] - 2s 102ms/step - loss: 0.9959 - accuracy: 0.5705
- val_loss: 1.1083 - val_accuracy: 0.4837
Epoch 5/20
20/20 [==============================] - 2s 103ms/step - loss: 0.8901 - accuracy: 0.6328
- val_loss: 1.1368 - val_accuracy: 0.5163
Epoch 6/20
20/20 [==============================] - 2s 109ms/step - loss: 0.8231 - accuracy: 0.6967
```

```
- val_loss: 1.0816 - val_accuracy: 0.5686
Epoch 7/20
20/20 [==============================] - 2s 123ms/step - loss: 0.7430 - accuracy: 0.7279
- val_loss: 1.0766 - val_accuracy: 0.5882
Epoch 8/20
20/20 [==============================] - 2s 124ms/step - loss: 0.7024 - accuracy: 0.7623
- val_loss: 0.9961 - val_accuracy: 0.5882
Epoch 9/20
20/20 [==============================] - 2s 116ms/step - loss: 0.6275 - accuracy: 0.7984
- val_loss: 1.0127 - val_accuracy: 0.5817
Epoch 10/20
20/20 [==============================] - 2s 120ms/step - loss: 0.5891 - accuracy: 0.8049
- val_loss: 1.1383 - val_accuracy: 0.5621
Epoch 11/20
20/20 [==============================] - 2s 118ms/step - loss: 0.5677 - accuracy: 0.8066
- val_loss: 1.0155 - val_accuracy: 0.5556
Epoch 12/20
20/20 [==============================] - 2s 111ms/step - loss: 0.5085 - accuracy: 0.8197
- val_loss: 1.0607 - val_accuracy: 0.5556
Epoch 13/20
20/20 [==============================] - 2s 119ms/step - loss: 0.4824 - accuracy: 0.8311
- val_loss: 1.1328 - val_accuracy: 0.5817
Epoch 14/20
20/20 [==============================] - 2s 119ms/step - loss: 0.4806 - accuracy: 0.8098
- val_loss: 1.0487 - val_accuracy: 0.5686
Epoch 15/20
20/20 [==============================] - 3s 132ms/step - loss: 0.4420 - accuracy: 0.8393
- val_loss: 1.0452 - val_accuracy: 0.5817
Epoch 16/20
20/20 [==============================] - 3s 169ms/step - loss: 0.3997 - accuracy: 0.8803
- val_loss: 1.0697 - val_accuracy: 0.5948
Epoch 17/20
20/20 [==============================] - 3s 134ms/step - loss: 0.3725 - accuracy: 0.8803
- val_loss: 1.0612 - val_accuracy: 0.5817
Epoch 18/20
20/20 [==============================] - 2s 117ms/step - loss: 0.3565 - accuracy: 0.9049
- val_loss: 1.0577 - val_accuracy: 0.5948
Epoch 19/20
20/20 [==============================] - 2s 112ms/step - loss: 0.3408 - accuracy: 0.8934
- val_loss: 1.0361 - val_accuracy: 0.6144
Epoch 20/20
20/20 [==============================] - 3s 136ms/step - loss: 0.3299 - accuracy: 0.9115
- val_loss: 1.0588 - val_accuracy: 0.5882
```

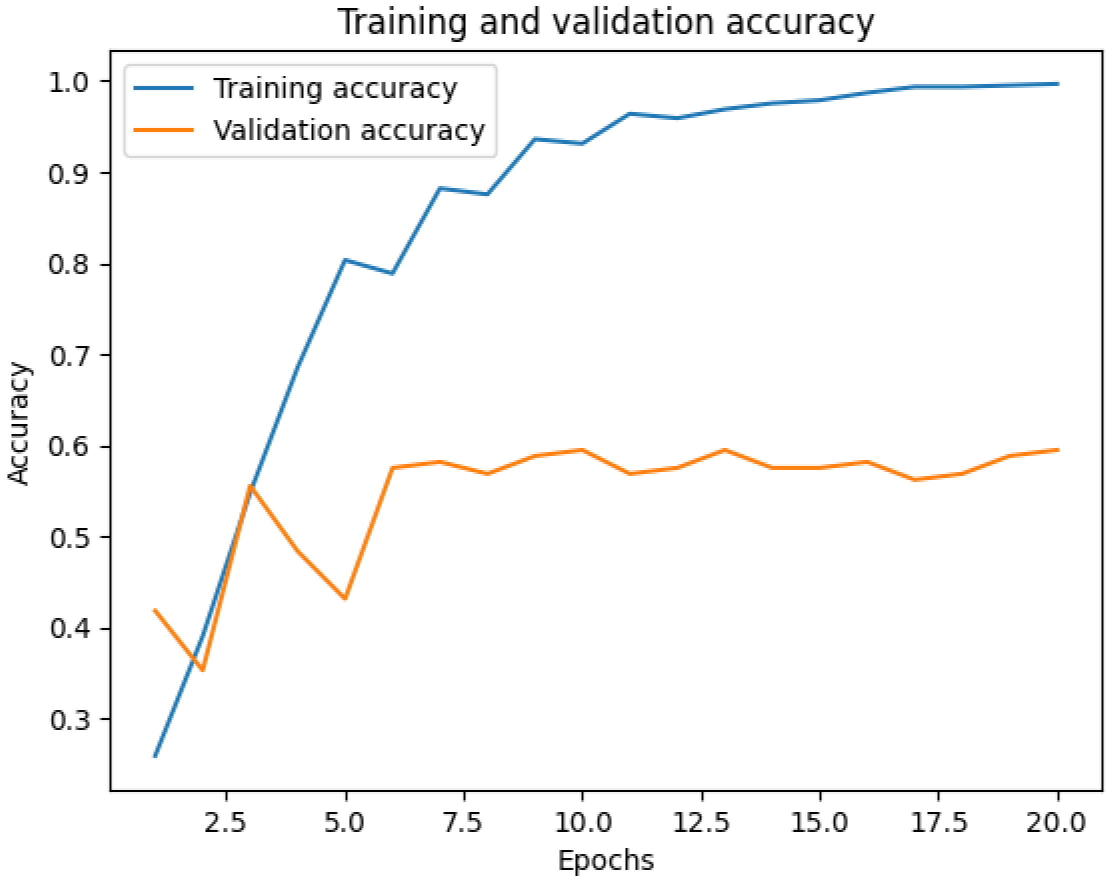


In [106…

```
updated_model = Sequential([
    Conv2D(8, (3, 3), activation='relu', input_shape=(image_height, image_width, 3)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(4, activation='softmax')
])

updated_model.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

training_history = updated_model.fit(X_train, y_train, epochs=20, validation_data=(X_val
plot_training_curves(training_history)
```

```
Epoch 1/20
20/20 [==============================] - 4s 130ms/step - loss: 2.1129 - accuracy: 0.2590
- val_loss: 1.3282 - val_accuracy: 0.4183
Epoch 2/20
20/20 [==============================] - 2s 119ms/step - loss: 1.2537 - accuracy: 0.3902
- val_loss: 1.3368 - val_accuracy: 0.3529
Epoch 3/20
20/20 [==============================] - 2s 110ms/step - loss: 1.0434 - accuracy: 0.5475
- val_loss: 1.1303 - val_accuracy: 0.5556
Epoch 4/20
20/20 [==============================] - 2s 109ms/step - loss: 0.8145 - accuracy: 0.6852
- val_loss: 1.0848 - val_accuracy: 0.4837
Epoch 5/20
20/20 [==============================] - 2s 108ms/step - loss: 0.6071 - accuracy: 0.8033
- val_loss: 1.2476 - val_accuracy: 0.4314
Epoch 6/20
20/20 [==============================] - 2s 109ms/step - loss: 0.5809 - accuracy: 0.7885
- val_loss: 0.9998 - val_accuracy: 0.5752
Epoch 7/20
20/20 [==============================] - 2s 116ms/step - loss: 0.4411 - accuracy: 0.8820
- val_loss: 1.0405 - val_accuracy: 0.5817
```

```
Epoch 8/20
20/20 [==============================] - 2s 107ms/step - loss: 0.4496 - accuracy: 0.8754
- val_loss: 1.0009 - val_accuracy: 0.5686
Epoch 9/20
20/20 [==============================] - 2s 108ms/step - loss: 0.3279 - accuracy: 0.9361
- val_loss: 1.0263 - val_accuracy: 0.5882
Epoch 10/20
20/20 [==============================] - 2s 106ms/step - loss: 0.2838 - accuracy: 0.9311
- val_loss: 0.9770 - val_accuracy: 0.5948
Epoch 11/20
20/20 [==============================] - 2s 108ms/step - loss: 0.2363 - accuracy: 0.9639
- val_loss: 0.9905 - val_accuracy: 0.5686
Epoch 12/20
20/20 [==============================] - 2s 111ms/step - loss: 0.2075 - accuracy: 0.9590
- val_loss: 1.0256 - val_accuracy: 0.5752
Epoch 13/20
20/20 [==============================] - 2s 108ms/step - loss: 0.1955 - accuracy: 0.9689
- val_loss: 1.0245 - val_accuracy: 0.5948
Epoch 14/20
20/20 [==============================] - 2s 121ms/step - loss: 0.1641 - accuracy: 0.9754
- val_loss: 1.0506 - val_accuracy: 0.5752
Epoch 15/20
20/20 [==============================] - 2s 112ms/step - loss: 0.1386 - accuracy: 0.9787
- val_loss: 1.0321 - val_accuracy: 0.5752
Epoch 16/20
20/20 [==============================] - 2s 108ms/step - loss: 0.1219 - accuracy: 0.9869
- val_loss: 1.0683 - val_accuracy: 0.5817
Epoch 17/20
20/20 [==============================] - 2s 109ms/step - loss: 0.0950 - accuracy: 0.9934
- val_loss: 1.0534 - val_accuracy: 0.5621
Epoch 18/20
20/20 [==============================] - 2s 107ms/step - loss: 0.0822 - accuracy: 0.9934
- val_loss: 1.0907 - val_accuracy: 0.5686
Epoch 19/20
20/20 [==============================] - 2s 109ms/step - loss: 0.0727 - accuracy: 0.9951
- val_loss: 1.0914 - val_accuracy: 0.5882
Epoch 20/20
20/20 [==============================] - 2s 107ms/step - loss: 0.0692 - accuracy: 0.9967
- val_loss: 1.0865 - val_accuracy: 0.5948
```


Training and validation accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0