dm-1-asst-3

November 25, 2023

[1]:
```
import pandas as pd
import seaborn as sns
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

1.Association Rule Generation from Transaction Data

[2]:
```
data = pd.read_csv('/kaggle/input/dataset/Grocery_Items_14.csv')
```

[3]:
```
Transaction = []
for i in range(data.shape[0]):
    Transaction.append([str(data.values[i,j]) for j in range(data.shape[1]) if␣
 ↪str(data.values[i,j]) != 'nan'])
```

[4]:
```
te = TransactionEncoder()
te_ary = te.fit(Transaction).transform(Transaction)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

(C)

[5]:
```
frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence",␣
 ↪min_threshold=0.1)
```

[6]:
```
rules
```

[6]:
```
           antecedents          consequents  antecedent support  \
0               (soda)  (other vegetables)            0.097250
1  (other vegetables)        (whole milk)            0.121250
2         (rolls/buns)        (whole milk)            0.109625
3               (soda)        (whole milk)            0.097250
4             (yogurt)        (whole milk)            0.085750

   consequent support   support  confidence      lift  leverage  conviction  \
0            0.121250  0.010250    0.105398  0.869266 -0.001542    0.982281
1            0.157875  0.014625    0.120619  0.764013 -0.004517    0.957633
2            0.157875  0.014500    0.132269  0.837809 -0.002807    0.970491
3            0.157875  0.011125    0.114396  0.724598 -0.004228    0.950905
```

```
4           0.157875  0.011625    0.135569  0.858708 -0.001913    0.974195

   zhangs_metric
0      -0.142807
1      -0.260080
2      -0.178594
3      -0.296280
4      -0.152523
```

(D)

[7]:
```python
msv = [0.001,0.005,0.01,0.05]

mct = [0.05,0.075,0.1]
```

[8]:
```python
h_data = []
for i in msv:
    for j in mct:
        frequent_itemsets = apriori(df, min_support=i, use_colnames=True)
        rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=j)
        res = len(rules)
        h_data.append({'msv': i, 'mct': j, 'rules_count': res})
```

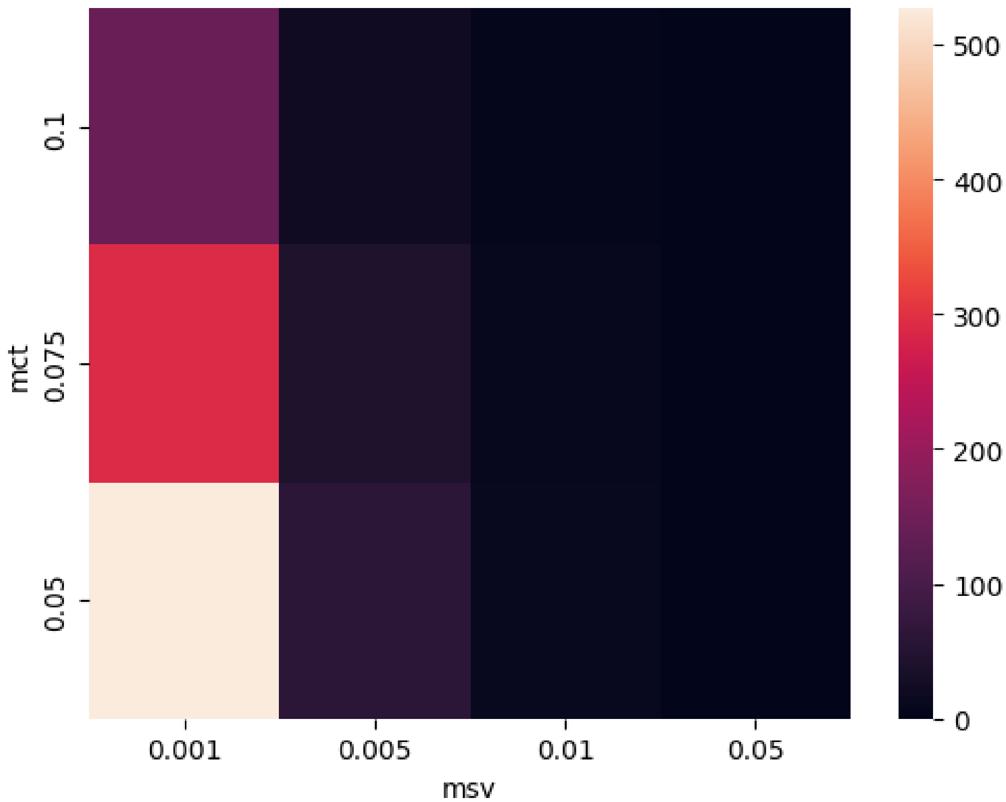
[9]:
```python
heatdata = pd.DataFrame(h_data)
heatdata = heatdata.pivot(index='mct', columns='msv', values='rules_count')
```

[10]:
```python
heatdata
```

[10]:
```
msv    0.001  0.005  0.010  0.050
mct
0.050    527     60     12      0
0.075    292     43     10      0
0.100    144     22      5      0
```

[11]:
```python
sns.heatmap(heatdata.sort_index(ascending=False))
```

[11]:
```
<Axes: xlabel='msv', ylabel='mct'>
```



(E)

[12]:
```
frequent_itemsets_2 = apriori(df, min_support=0.005, use_colnames=True)
rules_2 = association_rules(frequent_itemsets_2, metric="confidence",␣
↪min_threshold=0.0)
```

[13]:
```
rules_2.nlargest(1,'confidence')
```

[13]:
```
      antecedents          consequents  antecedent support  consequent support  \
18  (frankfurter)  (other vegetables)            0.038125             0.12125

     support  confidence      lift  leverage  conviction  zhangs_metric
18  0.006125    0.160656  1.324996  0.001502    1.046948       0.255003
```

2.Image Classification using CNN

[14]:
```
import os
import keras
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, Dense, MaxPool2D, Flatten, Dropout,␣
↪BatchNormalization, DepthwiseConv2D, InputLayer, GlobalAveragePooling2D
```

```
from tensorflow.keras.regularizers import l2
from tensorflow.keras.models import Sequential
```

[15]:
```
directory = "/kaggle/input/reena-dataset"
```

[16]:
```
def images_labels(data_directory):
    Images_data = []
    labels_data = []
    classes = os.listdir(data_directory)
    for i in classes:
        for j in os.listdir(os.path.join(data_directory,i)):
            Images_data.append(os.path.join(data_directory,i,j))
            labels_data.append(i)
    return([Images_data,labels_data])
```

[17]:
```
def data_split(data,labels,train_size):
    train_imgs,test_imgs,train_label,test_label = train_test_split(data,
                                                                labels,
                                                               ␣
 ↪train_size=train_size,
                                                                random_state=10)
    return([train_imgs,test_imgs,train_label,test_label])
```

[18]:
```
Images_data,labels_data = images_labels(directory)
train_imgs,test_imgs,train_label,test_label =␣
 ↪data_split(Images_data,labels_data,0.8)
```

[19]:
```
datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
```

[20]:
```
train_data = datagen.flow_from_dataframe(
    pd.DataFrame({'file_paths': train_imgs, 'labels': train_label}),
    x_col='file_paths',
    y_col='labels',
    target_size=(256, 256),
    batch_size=32,
    class_mode='categorical',
)
```

```
Found 527 validated image filenames belonging to 4 classes.
```

[21]:
```
test_data = datagen.flow_from_dataframe(
    pd.DataFrame({'file_paths': test_imgs, 'labels': test_label}),
    x_col='file_paths',
    y_col='labels',
    target_size=(256, 256),
    batch_size=32,
    class_mode='categorical',
```

```
)
```

```
Found 132 validated image filenames belonging to 4 classes.
```

[22]:
```
cnn_1 = Sequential([
    Conv2D(8, (3, 3), activation='relu', input_shape=(256, 256, 3)),
    MaxPool2D(2, 2),
    Flatten(),
    Dense(16, activation='relu'),
    Dense(4, activation='softmax')
])
cnn_1.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
cnn_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 8)       224

 max_pooling2d (MaxPooling2  (None, 127, 127, 8)       0
 D)

 flatten (Flatten)           (None, 129032)            0

 dense (Dense)               (None, 16)                2064528

 dense_1 (Dense)             (None, 4)                 68

=================================================================
Total params: 2064820 (7.88 MB)
Trainable params: 2064820 (7.88 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[23]:
```
train_1 = cnn_1.fit(train_data,
                    validation_data = test_data,
                    epochs = 20)
```

```
Epoch 1/20
17/17 [==============================] - 9s 472ms/step - loss: 1.9831 -
accuracy: 0.2448 - val_loss: 1.4567 - val_accuracy: 0.2424
Epoch 2/20
17/17 [==============================] - 8s 440ms/step - loss: 1.3392 -
accuracy: 0.3795 - val_loss: 1.3668 - val_accuracy: 0.2652
Epoch 3/20
```

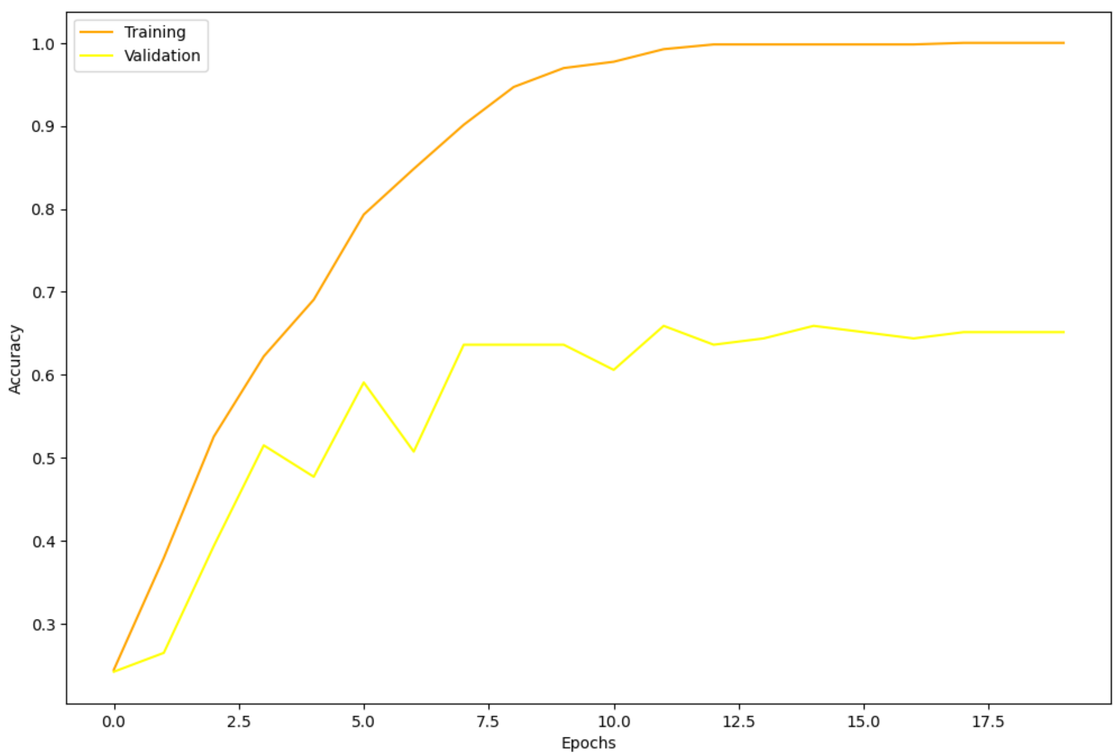
```
17/17 [==============================] - 7s 412ms/step - loss: 1.1690 -
accuracy: 0.5256 - val_loss: 1.2282 - val_accuracy: 0.3939
Epoch 4/20
17/17 [==============================] - 7s 417ms/step - loss: 0.9513 -
accuracy: 0.6224 - val_loss: 1.1976 - val_accuracy: 0.5152
Epoch 5/20
17/17 [==============================] - 7s 403ms/step - loss: 0.7637 -
accuracy: 0.6907 - val_loss: 1.3209 - val_accuracy: 0.4773
Epoch 6/20
17/17 [==============================] - 7s 408ms/step - loss: 0.6388 -
accuracy: 0.7932 - val_loss: 1.1764 - val_accuracy: 0.5909
Epoch 7/20
17/17 [==============================] - 7s 400ms/step - loss: 0.4496 -
accuracy: 0.8482 - val_loss: 1.2553 - val_accuracy: 0.5076
Epoch 8/20
17/17 [==============================] - 7s 397ms/step - loss: 0.3445 -
accuracy: 0.9013 - val_loss: 1.1815 - val_accuracy: 0.6364
Epoch 9/20
17/17 [==============================] - 8s 439ms/step - loss: 0.2317 -
accuracy: 0.9469 - val_loss: 1.1707 - val_accuracy: 0.6364
Epoch 10/20
17/17 [==============================] - 7s 404ms/step - loss: 0.1708 -
accuracy: 0.9696 - val_loss: 1.1616 - val_accuracy: 0.6364
Epoch 11/20
17/17 [==============================] - 7s 403ms/step - loss: 0.1193 -
accuracy: 0.9772 - val_loss: 1.2087 - val_accuracy: 0.6061
Epoch 12/20
17/17 [==============================] - 7s 402ms/step - loss: 0.0738 -
accuracy: 0.9924 - val_loss: 1.1887 - val_accuracy: 0.6591
Epoch 13/20
17/17 [==============================] - 7s 438ms/step - loss: 0.0484 -
accuracy: 0.9981 - val_loss: 1.2497 - val_accuracy: 0.6364
Epoch 14/20
17/17 [==============================] - 7s 438ms/step - loss: 0.0348 -
accuracy: 0.9981 - val_loss: 1.2645 - val_accuracy: 0.6439
Epoch 15/20
17/17 [==============================] - 7s 404ms/step - loss: 0.0275 -
accuracy: 0.9981 - val_loss: 1.2805 - val_accuracy: 0.6591
Epoch 16/20
17/17 [==============================] - 7s 397ms/step - loss: 0.0220 -
accuracy: 0.9981 - val_loss: 1.2952 - val_accuracy: 0.6515
Epoch 17/20
17/17 [==============================] - 8s 439ms/step - loss: 0.0176 -
accuracy: 0.9981 - val_loss: 1.3430 - val_accuracy: 0.6439
Epoch 18/20
17/17 [==============================] - 7s 424ms/step - loss: 0.0149 -
accuracy: 1.0000 - val_loss: 1.3086 - val_accuracy: 0.6515
Epoch 19/20
```

```
17/17 [==============================] - 7s 398ms/step - loss: 0.0123 -
accuracy: 1.0000 - val_loss: 1.3269 - val_accuracy: 0.6515
Epoch 20/20
17/17 [==============================] - 7s 401ms/step - loss: 0.0106 -
accuracy: 1.0000 - val_loss: 1.3416 - val_accuracy: 0.6515
```

[24]:

```python
accuracy = train_1.history['accuracy']
val_accuracy = train_1.history['val_accuracy']

epochs = range(len(accuracy))

fig = plt.figure(figsize=(12,8))
plt.plot(epochs,accuracy,c="orange",label="Training")
plt.plot(epochs,val_accuracy,c="yellow",label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
```

[24]: <matplotlib.legend.Legend at 0x7c35d8365e10>



Rowan Banner ID - 916462025

Hence b) Train the CNN using 2 other number of filters: 4 and 16 for the convolution layer (i) with all other parameters unchanged

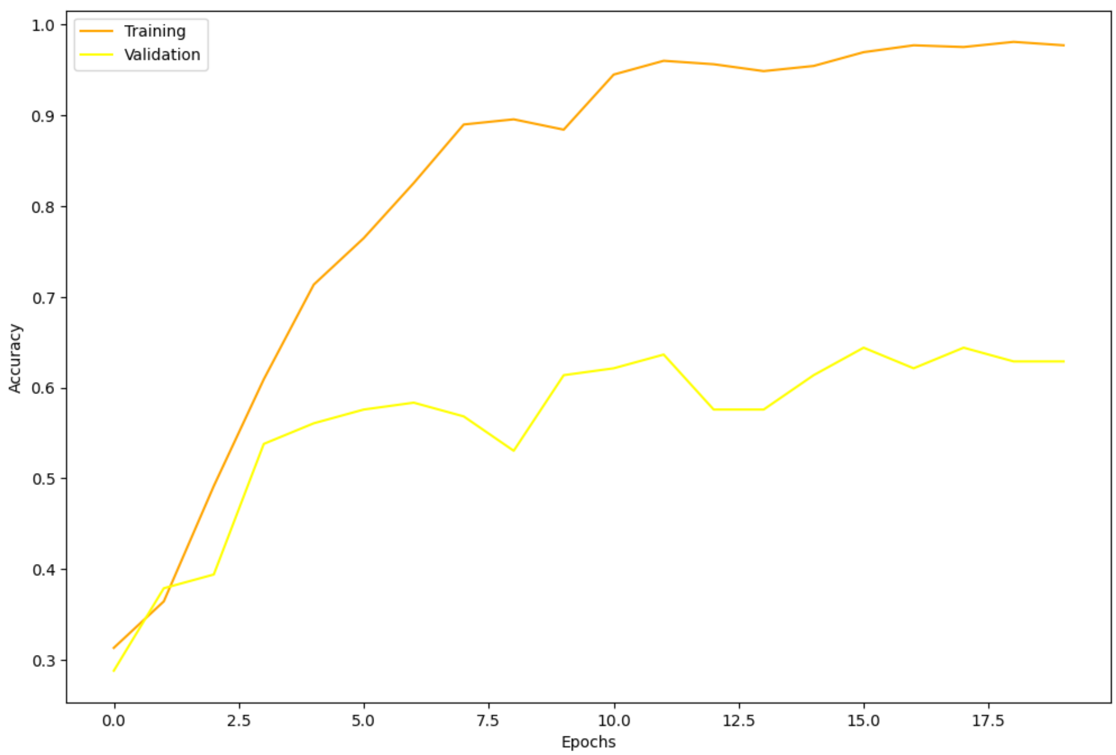
[25]:
```
cnn_2 = Sequential([
    Conv2D(4, (3, 3), activation='relu', input_shape=(256, 256, 3)),
    MaxPool2D(2, 2),
    Flatten(),
    Dense(16, activation='relu'),
    Dense(4, activation='softmax')
])
cnn_2.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
cnn_2.summary()

train_2 = cnn_2.fit(train_data,
                   validation_data = test_data,
                   epochs = 20)

accuracy = train_2.history['accuracy']
val_accuracy = train_2.history['val_accuracy']

epochs = range(len(accuracy))

fig = plt.figure(figsize=(12,8))
plt.plot(epochs,accuracy,c="orange",label="Training")
plt.plot(epochs,val_accuracy,c="yellow",label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 254, 254, 4)       112

 max_pooling2d_1 (MaxPoolin  (None, 127, 127, 4)       0
 g2D)

 flatten_1 (Flatten)         (None, 64516)             0

 dense_2 (Dense)             (None, 16)                1032272

 dense_3 (Dense)             (None, 4)                 68

=================================================================
Total params: 1032452 (3.94 MB)
Trainable params: 1032452 (3.94 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
----------------------------------------------------------------
Epoch 1/20
17/17 [==============================] - 8s 399ms/step - loss: 4.2745 -
accuracy: 0.3131 - val_loss: 2.5318 - val_accuracy: 0.2879
Epoch 2/20
17/17 [==============================] - 6s 347ms/step - loss: 1.4827 -
accuracy: 0.3643 - val_loss: 1.3328 - val_accuracy: 0.3788
Epoch 3/20
17/17 [==============================] - 6s 353ms/step - loss: 1.1432 -
accuracy: 0.4915 - val_loss: 1.2162 - val_accuracy: 0.3939
Epoch 4/20
17/17 [==============================] - 6s 355ms/step - loss: 0.9790 -
accuracy: 0.6091 - val_loss: 1.0551 - val_accuracy: 0.5379
Epoch 5/20
17/17 [==============================] - 6s 364ms/step - loss: 0.7697 -
accuracy: 0.7135 - val_loss: 1.1207 - val_accuracy: 0.5606
Epoch 6/20
17/17 [==============================] - 7s 387ms/step - loss: 0.6573 -
accuracy: 0.7647 - val_loss: 1.1818 - val_accuracy: 0.5758
Epoch 7/20
17/17 [==============================] - 6s 356ms/step - loss: 0.5208 -
accuracy: 0.8254 - val_loss: 1.0963 - val_accuracy: 0.5833
Epoch 8/20
17/17 [==============================] - 6s 352ms/step - loss: 0.4198 -
accuracy: 0.8899 - val_loss: 1.1906 - val_accuracy: 0.5682
Epoch 9/20
17/17 [==============================] - 6s 354ms/step - loss: 0.3960 -
accuracy: 0.8956 - val_loss: 1.3541 - val_accuracy: 0.5303
Epoch 10/20
17/17 [==============================] - 6s 362ms/step - loss: 0.3809 -
accuracy: 0.8843 - val_loss: 1.2113 - val_accuracy: 0.6136
Epoch 11/20
17/17 [==============================] - 6s 357ms/step - loss: 0.2910 -
accuracy: 0.9450 - val_loss: 1.1751 - val_accuracy: 0.6212
Epoch 12/20
17/17 [==============================] - 6s 360ms/step - loss: 0.2267 -
accuracy: 0.9602 - val_loss: 1.1755 - val_accuracy: 0.6364
Epoch 13/20
17/17 [==============================] - 6s 369ms/step - loss: 0.2292 -
accuracy: 0.9564 - val_loss: 1.2655 - val_accuracy: 0.5758
Epoch 14/20
17/17 [==============================] - 7s 393ms/step - loss: 0.2311 -
accuracy: 0.9488 - val_loss: 1.2992 - val_accuracy: 0.5758
Epoch 15/20
17/17 [==============================] - 6s 353ms/step - loss: 0.2067 -
accuracy: 0.9545 - val_loss: 1.2051 - val_accuracy: 0.6136
Epoch 16/20
17/17 [==============================] - 6s 356ms/step - loss: 0.1605 -
```

```
accuracy: 0.9696 - val_loss: 1.2384 - val_accuracy: 0.6439
Epoch 17/20
17/17 [==============================] - 6s 359ms/step - loss: 0.1400 -
accuracy: 0.9772 - val_loss: 1.2334 - val_accuracy: 0.6212
Epoch 18/20
17/17 [==============================] - 6s 352ms/step - loss: 0.1347 -
accuracy: 0.9753 - val_loss: 1.2296 - val_accuracy: 0.6439
Epoch 19/20
17/17 [==============================] - 6s 354ms/step - loss: 0.1212 -
accuracy: 0.9810 - val_loss: 1.2464 - val_accuracy: 0.6288
Epoch 20/20
17/17 [==============================] - 7s 391ms/step - loss: 0.1064 -
accuracy: 0.9772 - val_loss: 1.2823 - val_accuracy: 0.6288
```

[25]: <matplotlib.legend.Legend at 0x7c35bcdaf2b0>



[26]:
```
cnn_3 = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=(256, 256, 3)),
    MaxPool2D(2, 2),
    Flatten(),
    Dense(16, activation='relu'),
    Dense(4, activation='softmax')
])
```

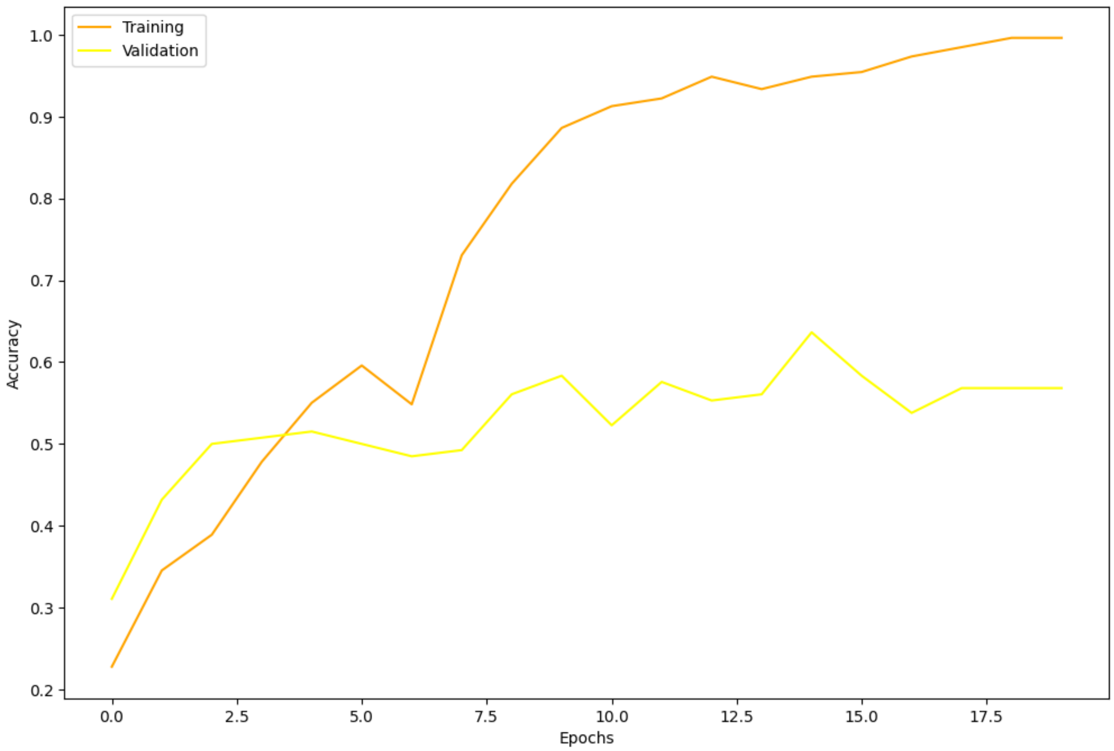
```
cnn_3.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
cnn_3.summary()

train_3 = cnn_3.fit(train_data,
                   validation_data = test_data,
                   epochs = 20)

accuracy = train_3.history['accuracy']
val_accuracy = train_3.history['val_accuracy']

epochs = range(len(accuracy))

fig = plt.figure(figsize=(12,8))
plt.plot(epochs,accuracy,c="orange",label="Training")
plt.plot(epochs,val_accuracy,c="yellow",label="Validation")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 254, 254, 16)      448

 max_pooling2d_2 (MaxPoolin  (None, 127, 127, 16)      0
 g2D)

 flatten_2 (Flatten)         (None, 258064)            0

 dense_4 (Dense)             (None, 16)                4129040

 dense_5 (Dense)             (None, 4)                 68

=================================================================
Total params: 4129556 (15.75 MB)
Trainable params: 4129556 (15.75 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
17/17 [==============================] - 11s 590ms/step - loss: 9.6415 -
accuracy: 0.2277 - val_loss: 2.3894 - val_accuracy: 0.3106
Epoch 2/20
17/17 [==============================] - 10s 561ms/step - loss: 1.6883 -
accuracy: 0.3454 - val_loss: 1.4788 - val_accuracy: 0.4318
```

```
Epoch 3/20
17/17 [==============================] - 10s 609ms/step - loss: 1.3059 -
accuracy: 0.3890 - val_loss: 1.1388 - val_accuracy: 0.5000
Epoch 4/20
17/17 [==============================] - 10s 566ms/step - loss: 1.0517 -
accuracy: 0.4782 - val_loss: 1.0898 - val_accuracy: 0.5076
Epoch 5/20
17/17 [==============================] - 10s 563ms/step - loss: 0.9316 -
accuracy: 0.5503 - val_loss: 1.1747 - val_accuracy: 0.5152
Epoch 6/20
17/17 [==============================] - 10s 594ms/step - loss: 0.8632 -
accuracy: 0.5958 - val_loss: 1.1144 - val_accuracy: 0.5000
Epoch 7/20
17/17 [==============================] - 10s 569ms/step - loss: 0.8302 -
accuracy: 0.5484 - val_loss: 1.2669 - val_accuracy: 0.4848
Epoch 8/20
17/17 [==============================] - 10s 587ms/step - loss: 0.6383 -
accuracy: 0.7306 - val_loss: 1.5191 - val_accuracy: 0.4924
Epoch 9/20
17/17 [==============================] - 10s 569ms/step - loss: 0.5095 -
accuracy: 0.8178 - val_loss: 1.1894 - val_accuracy: 0.5606
Epoch 10/20
17/17 [==============================] - 10s 588ms/step - loss: 0.3937 -
accuracy: 0.8861 - val_loss: 1.1720 - val_accuracy: 0.5833
Epoch 11/20
17/17 [==============================] - 10s 565ms/step - loss: 0.3391 -
accuracy: 0.9127 - val_loss: 1.3769 - val_accuracy: 0.5227
Epoch 12/20
17/17 [==============================] - 10s 595ms/step - loss: 0.2662 -
accuracy: 0.9222 - val_loss: 1.2057 - val_accuracy: 0.5758
Epoch 13/20
17/17 [==============================] - 10s 576ms/step - loss: 0.2352 -
accuracy: 0.9488 - val_loss: 1.4436 - val_accuracy: 0.5530
Epoch 14/20
17/17 [==============================] - 10s 576ms/step - loss: 0.2130 -
accuracy: 0.9336 - val_loss: 1.4518 - val_accuracy: 0.5606
Epoch 15/20
17/17 [==============================] - 10s 603ms/step - loss: 0.1983 -
accuracy: 0.9488 - val_loss: 1.2200 - val_accuracy: 0.6364
Epoch 16/20
17/17 [==============================] - 10s 572ms/step - loss: 0.1809 -
accuracy: 0.9545 - val_loss: 1.4571 - val_accuracy: 0.5833
Epoch 17/20
17/17 [==============================] - 10s 614ms/step - loss: 0.1189 -
accuracy: 0.9734 - val_loss: 1.6132 - val_accuracy: 0.5379
Epoch 18/20
17/17 [==============================] - 10s 577ms/step - loss: 0.0852 -
accuracy: 0.9848 - val_loss: 1.5012 - val_accuracy: 0.5682
```

```
Epoch 19/20
17/17 [==============================] - 10s 604ms/step - loss: 0.0700 -
accuracy: 0.9962 - val_loss: 1.5129 - val_accuracy: 0.5682
Epoch 20/20
17/17 [==============================] - 10s 582ms/step - loss: 0.0489 -
accuracy: 0.9962 - val_loss: 1.5818 - val_accuracy: 0.5682
```

[26]: <matplotlib.legend.Legend at 0x7c35bca1bd60>



On comparing the three models' accuracy graphs, cnn_1 is mostly ideal since it has reached an stable state after certain epochs, yet training accuracy is very high compared to validation accuracy so it also might be overfitting, same goes for cnn_2 model. where as the cnn_3 model is undefitting at first and overfitting later.