Connecting to Google Drive through mount point setup

```
from google.colab import drive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
drive.flush_and_unmount()
```

```
drive.mount('/content/drive')
ROOT = "/content/drive/"
```

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

```
file_path = ROOT+"MyDrive/Data_Mining/Grocery_Items_64.csv"
```

```
import os
import pandas as pd
import numpy as np
import sys
```

Loading the File into pandas DataFrame

```
df = pd.read_csv(file_path)
```

Applying the Transaction Encoder on Loaded dataset

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth,association_rules
import warnings
warnings.simplefilter(action="ignore", category=DeprecationWarning)
```

```
te = TransactionEncoder()
df = df.fillna("empty_item")
t_ary = te.fit_transform(df.values.tolist())
df = pd.DataFrame(t_ary, columns=te.columns_)
```

```
if "empty_item" in df.columns.tolist():
  df = df.drop(["empty_item"],axis=1)
else:
  print("Empty_item column is not created")
```

Extracting the Frequent Itemsets using FPMax approach

URL reference: https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/

```
frequent_itemset = fpgrowth(df,min_support=0.01,use_colnames=True)
extracted_rules = association_rules(frequent_itemset,metric="confidence",min_threshold=0.1)
```

```
extracted_rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(rolls/buns)	(whole milk)	0.108375	0.157375	0.015125	0.139562	0.886810	-0.001931	0.979297	-0.125225
1	(other vegetables)	(whole milk)	0.121500	0.157375	0.015000	0.123457	0.784475	-0.004121	0.961305	-0.238232
2	(yogurt)	(whole milk)	0.090875	0.157375	0.011500	0.126547	0.804114	-0.002801	0.964706	-0.211328
3	(soda)	(whole milk)	0.093500	0.157375	0.011375	0.121658	0.773044	-0.003340	0.959336	-0.244639

Extracting the rules based on given Support and threshold values

```
support_values = [0.001, 0.005, 0.01, 0.05]
confidence_values = [0.05, 0.075, 0.1]
all_rules_count = []
for min_support in support_values:
    support_result = []
    frequent_itemsets = fpgrowth(df, min_support=min_support, use_colnames=True)
    for min_confidence in confidence_values:
        extracted_rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)
        support_result.append(len(extracted_rules))
    all_rules_count.append(support_result)
```

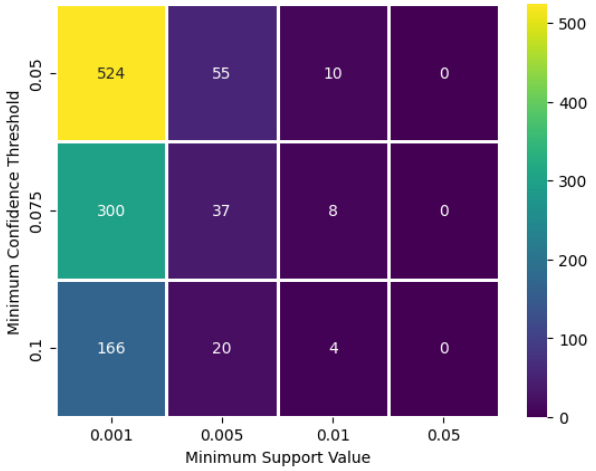
```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

Displaying the Extracted Rules count in the form of HeatMap

```
all_rules_count = np.array(all_rules_count)
transposed_data = all_rules_count.T
sns.heatmap(transposed_data,annot=True,linewidths=1,fmt='d',cmap='viridis')
plt.ylabel('Minimum Confidence Threshold')
plt.xlabel('Minimum Support Value')
plt.xticks(np.arange(len(support_values)) + 0.5, support_values)
plt.yticks(np.arange(len(confidence_values)) + 0.5, confidence_values)
```

```
([<matplotlib.axis.YTick at 0x79fbfa7cffd0>,
  <matplotlib.axis.YTick at 0x79fbfa7e8580>,
  <matplotlib.axis.YTick at 0x79fbfa63c130>],
 [Text(0, 0.5, '0.05'), Text(0, 1.5, '0.075'), Text(0, 2.5, '0.1')])
```



Double-click (or enter) to edit

```
min_support_threshold = 0.005
min_confidence_threshold = 0
frequent_itemset = fpgrowth(df, min_support=min_support_threshold, use_colnames=True)
extracted_rules = association_rules(frequent_itemset, metric='confidence', min_threshold=min_confidence_threshold)
max_confidence_value = max(extracted_rules['confidence'].tolist())
max_confidence_rules = extracted_rules[extracted_rules['confidence'] == max_confidence_value]
```

```
max_confidence_rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
48	(domestic eggs)	(whole milk)	0.039	0.157375	0.006125	0.157051	0.997943	-0.000013	0.999616	-0.00214

Maximum confidence values extracted was 0.157051

My Banner ID 916433420

```
from tensorflow import keras
from tensorflow.keras import layers
```

```
batch_size = 30 #considering the batch size as 30
```

```
images_location = ROOT+"MyDrive/Data_Mining/parent_folder_images_datasets"
```

```
training_images = keras.utils.image_dataset_from_directory(images_location,validation_split=0.2,subset="training",seed=123,image_size=(100,100),batch_size=batch_s
```

```
Found 760 files belonging to 4 classes.
Using 608 files for training.
```

```
testing_images =keras.utils.image_dataset_from_directory(images_location,validation_split=0.2,subset="validation",seed=123,image_size=(100,100),batch_size=batch_s
```

```
Found 760 files belonging to 4 classes.
Using 152 files for validation.
```

```
training_images = training_images.map(lambda x,y: (x/255, y))
testing_images = testing_images.map(lambda x,y: (x/255, y))
```

```
model = keras.Sequential([
            keras.Input(shape=(100,100,3)),
            layers.Conv2D(8, kernel_size=(3, 3), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
```

```
            layers.Dense(16, activation="relu"),
            layers.Dense(4, activation="softmax"),
            ])
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

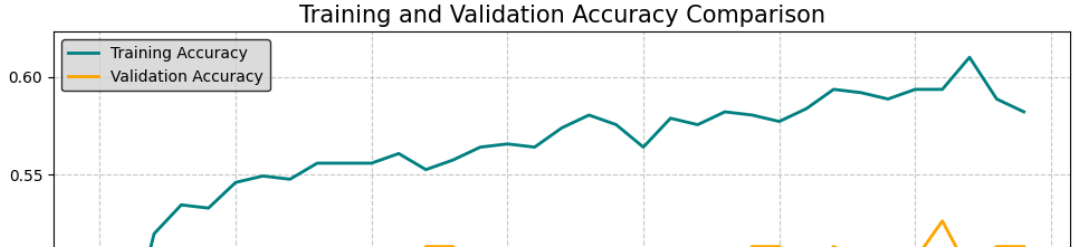
Reference

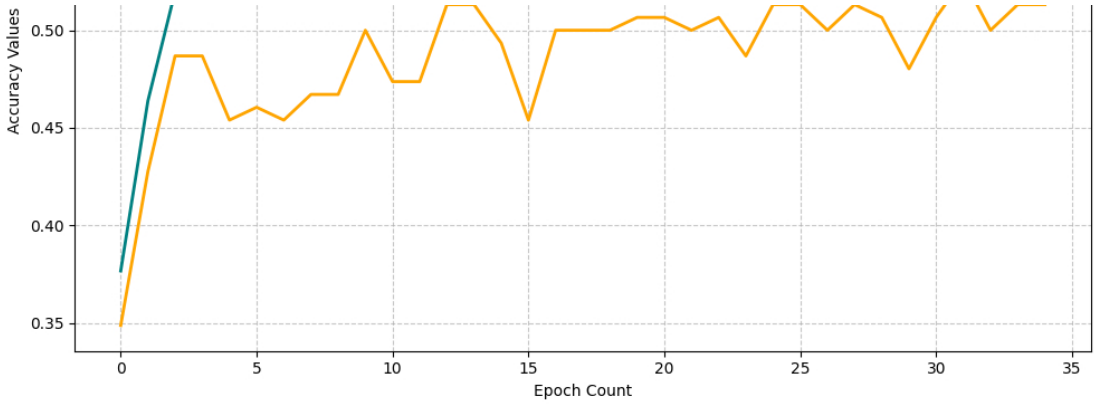
https://keras.io/examples/vision/mnist_convnet/

```
[ ] fitting_performance_first= model.fit(training_images , validation_data=testing_images, epochs=35,batch_size=25)
```

```
Epoch 7/35
21/21 [==============================] - 3s 122ms/step - loss: 1.0209 - accuracy: 0.5493 - val_loss: 1.1802 - val_accuracy: 0.4539
Epoch 8/35
21/21 [==============================] - 3s 109ms/step - loss: 1.0165 - accuracy: 0.5477 - val_loss: 1.1292 - val_accuracy: 0.4671
Epoch 9/35
21/21 [==============================] - 4s 186ms/step - loss: 0.9966 - accuracy: 0.5559 - val_loss: 1.1189 - val_accuracy: 0.4671
Epoch 10/35
21/21 [==============================] - 4s 149ms/step - loss: 0.9829 - accuracy: 0.5559 - val_loss: 1.0955 - val_accuracy: 0.5000
Epoch 11/35
21/21 [==============================] - 3s 125ms/step - loss: 0.9762 - accuracy: 0.5559 - val_loss: 1.0972 - val_accuracy: 0.4737
Epoch 12/35
21/21 [==============================] - 3s 145ms/step - loss: 0.9635 - accuracy: 0.5609 - val_loss: 1.1356 - val_accuracy: 0.4737
Epoch 13/35
21/21 [==============================] - 5s 194ms/step - loss: 0.9765 - accuracy: 0.5526 - val_loss: 1.0805 - val_accuracy: 0.5132
Epoch 14/35
21/21 [==============================] - 3s 111ms/step - loss: 0.9617 - accuracy: 0.5576 - val_loss: 1.0757 - val_accuracy: 0.5132
Epoch 15/35
21/21 [==============================] - 3s 107ms/step - loss: 0.9540 - accuracy: 0.5641 - val_loss: 1.0917 - val_accuracy: 0.4934
Epoch 16/35
21/21 [==============================] - 3s 125ms/step - loss: 0.9763 - accuracy: 0.5658 - val_loss: 1.2068 - val_accuracy: 0.4539
Epoch 17/35
21/21 [==============================] - 3s 122ms/step - loss: 0.9503 - accuracy: 0.5641 - val_loss: 1.0826 - val_accuracy: 0.5000
Epoch 18/35
21/21 [==============================] - 4s 163ms/step - loss: 0.9286 - accuracy: 0.5740 - val_loss: 1.0768 - val_accuracy: 0.5000
Epoch 19/35
21/21 [==============================] - 3s 108ms/step - loss: 0.9251 - accuracy: 0.5806 - val_loss: 1.1882 - val_accuracy: 0.5000
Epoch 20/35
21/21 [==============================] - 3s 110ms/step - loss: 0.9210 - accuracy: 0.5757 - val_loss: 1.0609 - val_accuracy: 0.5066
Epoch 21/35
21/21 [==============================] - 3s 144ms/step - loss: 0.9223 - accuracy: 0.5641 - val_loss: 1.0624 - val_accuracy: 0.5066
Epoch 22/35
21/21 [==============================] - 4s 182ms/step - loss: 0.9198 - accuracy: 0.5789 - val_loss: 1.1392 - val_accuracy: 0.5000
Epoch 23/35
21/21 [==============================] - 4s 184ms/step - loss: 0.9054 - accuracy: 0.5757 - val_loss: 1.0648 - val_accuracy: 0.5066
Epoch 24/35
21/21 [==============================] - 3s 107ms/step - loss: 0.9060 - accuracy: 0.5822 - val_loss: 1.1557 - val_accuracy: 0.4868
Epoch 25/35
21/21 [==============================] - 3s 124ms/step - loss: 0.8886 - accuracy: 0.5806 - val_loss: 1.1221 - val_accuracy: 0.5132
Epoch 26/35
21/21 [==============================] - 4s 183ms/step - loss: 0.8926 - accuracy: 0.5773 - val_loss: 1.0507 - val_accuracy: 0.5132
Epoch 27/35
21/21 [==============================] - 4s 180ms/step - loss: 0.8763 - accuracy: 0.5839 - val_loss: 1.0601 - val_accuracy: 0.5000
Epoch 28/35
21/21 [==============================] - 3s 108ms/step - loss: 0.8645 - accuracy: 0.5938 - val_loss: 1.1339 - val_accuracy: 0.5132
Epoch 29/35
21/21 [==============================] - 3s 110ms/step - loss: 0.8593 - accuracy: 0.5921 - val_loss: 1.0843 - val_accuracy: 0.5066
Epoch 30/35
21/21 [==============================] - 4s 157ms/step - loss: 0.8584 - accuracy: 0.5888 - val_loss: 1.0471 - val_accuracy: 0.4803
Epoch 31/35
21/21 [==============================] - 4s 186ms/step - loss: 0.8722 - accuracy: 0.5938 - val_loss: 1.1015 - val_accuracy: 0.5066
Epoch 32/35
21/21 [==============================] - 3s 106ms/step - loss: 0.8464 - accuracy: 0.5938 - val_loss: 1.1362 - val_accuracy: 0.5263
Epoch 33/35
21/21 [==============================] - 3s 106ms/step - loss: 0.8462 - accuracy: 0.6102 - val_loss: 1.2226 - val_accuracy: 0.5000
Epoch 34/35
21/21 [==============================] - 3s 134ms/step - loss: 0.8676 - accuracy: 0.5888 - val_loss: 1.0891 - val_accuracy: 0.5132
Epoch 35/35
21/21 [==============================] - 4s 188ms/step - loss: 0.8352 - accuracy: 0.5822 - val_loss: 1.0548 - val_accuracy: 0.5132
```

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(fitting_performance_first.history['accuracy'], color='teal', label='Training Accuracy', linewidth=2)
ax.plot(fitting_performance_first.history['val_accuracy'], color='orange', label='Validation Accuracy', linewidth=2)
ax.set_title('Training and Validation Accuracy Comparison', fontsize=15)
ax.set_ylabel('Accuracy Values')
ax.set_xlabel('Epoch Count')
ax.grid(True, linestyle='--', alpha=0.7)
legend = ax.legend(loc='upper left', frameon=True, edgecolor='black')
legend.get_texts()[0].set_text('Training Accuracy')
legend.get_texts()[1].set_text('Validation Accuracy')
legend.get_frame().set_facecolor('lightgray')
plt.tight_layout()
plt.show()
```





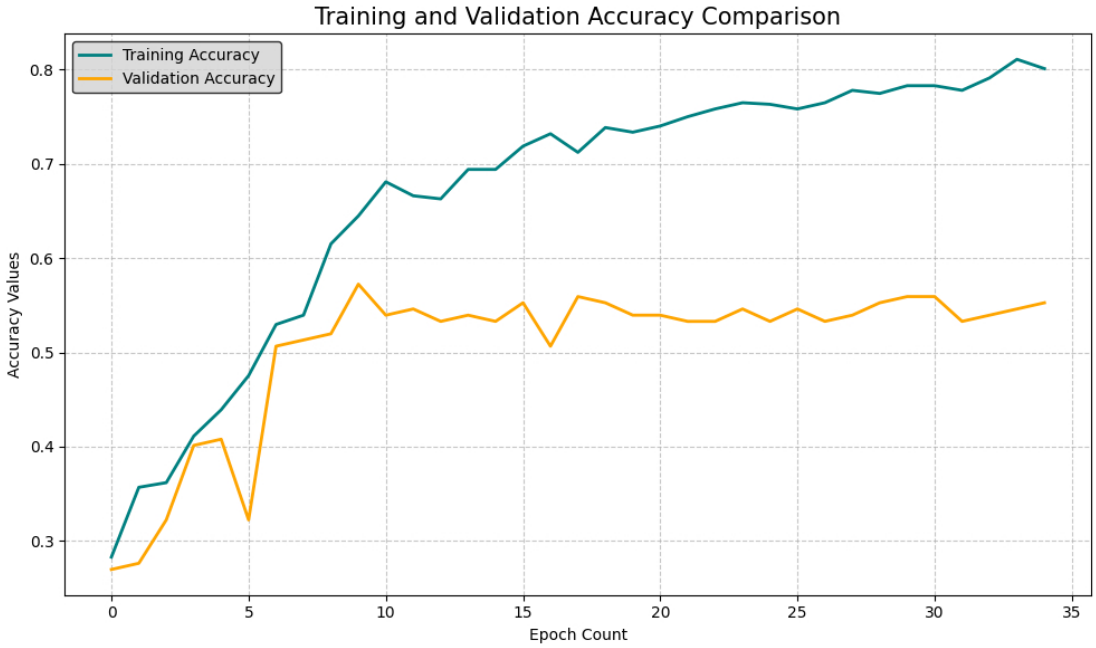

```
model = keras.Sequential([
          keras.Input(shape=(100,100,3)),
          layers.Conv2D(8, kernel_size=(5, 5), activation="relu"),
          layers.MaxPooling2D(pool_size=(2, 2)),
          layers.Flatten(),
          layers.Dense(16, activation="relu"),
          layers.Dense(4, activation="softmax"),
          ])
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
[ ] fitting_performance_second = model.fit(training_images , validation_data=testing_images, epochs=35,batch_size=25)
```

```
Epoch 1/35
21/21 [==============================] - 5s 164ms/step - loss: 1.8112 - accuracy: 0.2829 - val_loss: 1.3725 - val_accuracy: 0.2697
Epoch 2/35
21/21 [==============================] - 4s 154ms/step - loss: 1.3332 - accuracy: 0.3569 - val_loss: 1.3213 - val_accuracy: 0.2763
Epoch 3/35
21/21 [==============================] - 5s 205ms/step - loss: 1.2818 - accuracy: 0.3618 - val_loss: 1.2971 - val_accuracy: 0.3224
Epoch 4/35
21/21 [==============================] - 5s 237ms/step - loss: 1.2267 - accuracy: 0.4112 - val_loss: 1.2376 - val_accuracy: 0.4013
Epoch 5/35
21/21 [==============================] - 4s 169ms/step - loss: 1.1647 - accuracy: 0.4391 - val_loss: 1.1994 - val_accuracy: 0.4079
Epoch 6/35
21/21 [==============================] - 6s 248ms/step - loss: 1.1073 - accuracy: 0.4753 - val_loss: 1.1982 - val_accuracy: 0.3224
Epoch 7/35
21/21 [==============================] - 4s 169ms/step - loss: 1.0389 - accuracy: 0.5296 - val_loss: 1.1743 - val_accuracy: 0.5066
Epoch 8/35
21/21 [==============================] - 6s 274ms/step - loss: 1.0226 - accuracy: 0.5395 - val_loss: 1.0948 - val_accuracy: 0.5132
Epoch 9/35
21/21 [==============================] - 4s 178ms/step - loss: 0.9038 - accuracy: 0.6151 - val_loss: 1.1300 - val_accuracy: 0.5197
Epoch 10/35
21/21 [==============================] - 4s 169ms/step - loss: 0.8346 - accuracy: 0.6447 - val_loss: 1.0592 - val_accuracy: 0.5724
Epoch 11/35
21/21 [==============================] - 5s 225ms/step - loss: 0.7901 - accuracy: 0.6809 - val_loss: 1.0734 - val_accuracy: 0.5395
Epoch 12/35
21/21 [==============================] - 5s 223ms/step - loss: 0.7636 - accuracy: 0.6661 - val_loss: 1.0621 - val_accuracy: 0.5461
Epoch 13/35
21/21 [==============================] - 4s 158ms/step - loss: 0.7953 - accuracy: 0.6628 - val_loss: 1.0461 - val_accuracy: 0.5329
Epoch 14/35
21/21 [==============================] - 4s 159ms/step - loss: 0.7172 - accuracy: 0.6941 - val_loss: 1.0633 - val_accuracy: 0.5395
Epoch 15/35
21/21 [==============================] - 5s 239ms/step - loss: 0.7114 - accuracy: 0.6941 - val_loss: 1.0536 - val_accuracy: 0.5329
Epoch 16/35
21/21 [==============================] - 4s 171ms/step - loss: 0.6532 - accuracy: 0.7188 - val_loss: 1.0861 - val_accuracy: 0.5526
Epoch 17/35
21/21 [==============================] - 5s 224ms/step - loss: 0.6199 - accuracy: 0.7319 - val_loss: 1.1329 - val_accuracy: 0.5066
Epoch 18/35
21/21 [==============================] - 6s 238ms/step - loss: 0.6606 - accuracy: 0.7122 - val_loss: 1.0524 - val_accuracy: 0.5592
Epoch 19/35
21/21 [==============================] - 4s 155ms/step - loss: 0.6015 - accuracy: 0.7385 - val_loss: 1.1095 - val_accuracy: 0.5526
Epoch 20/35
21/21 [==============================] - 4s 157ms/step - loss: 0.6133 - accuracy: 0.7336 - val_loss: 1.0902 - val_accuracy: 0.5395
Epoch 21/35
21/21 [==============================] - 5s 235ms/step - loss: 0.5665 - accuracy: 0.7401 - val_loss: 1.1177 - val_accuracy: 0.5395
Epoch 22/35
21/21 [==============================] - 4s 170ms/step - loss: 0.5409 - accuracy: 0.7500 - val_loss: 1.1327 - val_accuracy: 0.5329
Epoch 23/35
21/21 [==============================] - 5s 221ms/step - loss: 0.5160 - accuracy: 0.7582 - val_loss: 1.1502 - val_accuracy: 0.5329
Epoch 24/35
21/21 [==============================] - 6s 239ms/step - loss: 0.5020 - accuracy: 0.7648 - val_loss: 1.1490 - val_accuracy: 0.5461
Epoch 25/35
21/21 [==============================] - 4s 159ms/step - loss: 0.4963 - accuracy: 0.7632 - val_loss: 1.1677 - val_accuracy: 0.5329
Epoch 26/35
21/21 [==============================] - 4s 168ms/step - loss: 0.4897 - accuracy: 0.7582 - val_loss: 1.1388 - val_accuracy: 0.5461
Epoch 27/35
21/21 [==============================] - 6s 269ms/step - loss: 0.4712 - accuracy: 0.7648 - val_loss: 1.1598 - val_accuracy: 0.5329
Epoch 28/35
21/21 [==============================] - 4s 158ms/step - loss: 0.4430 - accuracy: 0.7780 - val_loss: 1.1830 - val_accuracy: 0.5395
Epoch 29/35
21/21 [==============================] - 4s 173ms/step - loss: 0.4399 - accuracy: 0.7747 - val_loss: 1.2150 - val_accuracy: 0.5526
```

```
[ ]
fig, ax = plt.subplots(figsize=(10, 6))  # Adjust the figure size
ax.plot(fitting_performance_second.history['accuracy'], color='teal', label='Training Accuracy', linewidth=2)
ax.plot(fitting_performance_second.history['val_accuracy'], color='orange', label='Validation Accuracy', linewidth=2)
ax.set_title('Training and Validation Accuracy Comparison', fontsize=15)
ax.set_ylabel('Accuracy Values')
ax.set_xlabel('Epoch Count')
```

```
ax.grid(True, linestyle='--', alpha=0.7)
legend = ax.legend(loc='upper left', frameon=True, edgecolor='black')
legend.get_texts()[0].set_text('Training Accuracy')
legend.get_texts()[1].set_text('Validation Accuracy')
legend.get_frame().set_facecolor('lightgray')
plt.tight_layout()
plt.show()
```




```
[ ] model = keras.Sequential([
            keras.Input(shape=(100,100,3)),
            layers.Conv2D(8, kernel_size=(7, 7), activation="relu"),
            layers.MaxPooling2D(pool_size=(2, 2)),
            layers.Flatten(),
            layers.Dense(16, activation="relu"),
            layers.Dense(4, activation="softmax"),
            ])
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

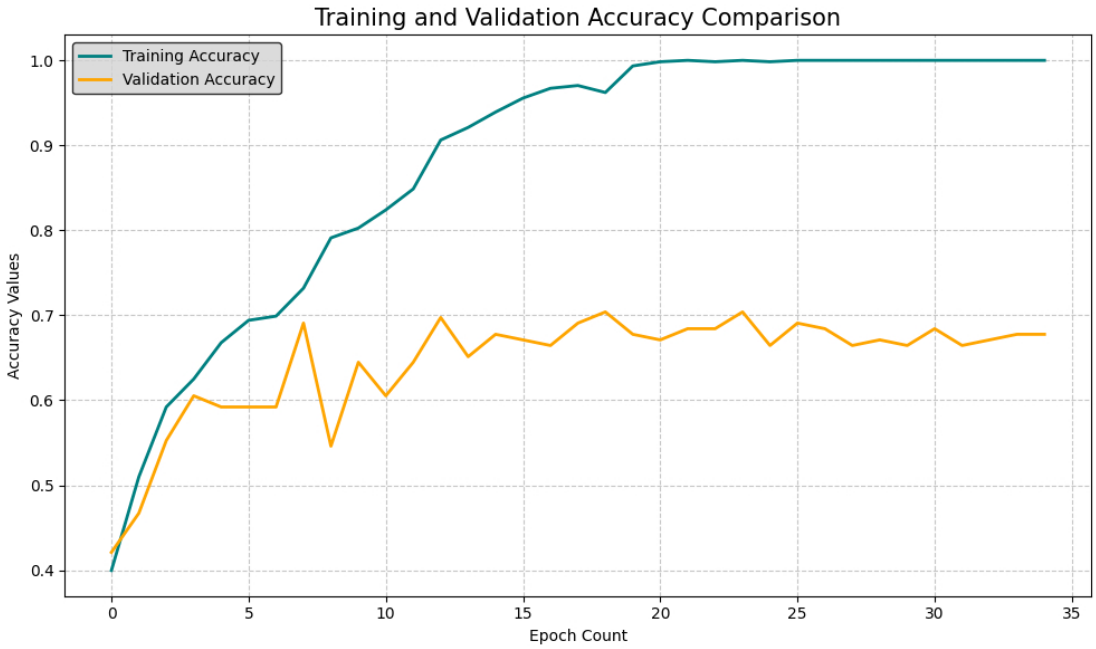
```
[ ] fitting_performance_thrid = model.fit(training_images , validation_data=testing_images, epochs=35,batch_size=25)
```

```
Epoch 1/35
21/21 [==============================] - 9s 357ms/step - loss: 1.3063 - accuracy: 0.3997 - val_loss: 1.1898 - val_accuracy: 0.4211
Epoch 2/35
21/21 [==============================] - 5s 237ms/step - loss: 1.0571 - accuracy: 0.5099 - val_loss: 1.0520 - val_accuracy: 0.4671
Epoch 3/35
21/21 [==============================] - 7s 307ms/step - loss: 0.9006 - accuracy: 0.5921 - val_loss: 0.9320 - val_accuracy: 0.5526
Epoch 4/35
21/21 [==============================] - 5s 238ms/step - loss: 0.8062 - accuracy: 0.6250 - val_loss: 0.8641 - val_accuracy: 0.6053
Epoch 5/35
21/21 [==============================] - 6s 261ms/step - loss: 0.6996 - accuracy: 0.6678 - val_loss: 0.8607 - val_accuracy: 0.5921
Epoch 6/35
21/21 [==============================] - 5s 238ms/step - loss: 0.6516 - accuracy: 0.6941 - val_loss: 0.8592 - val_accuracy: 0.5921
Epoch 7/35
21/21 [==============================] - 5s 237ms/step - loss: 0.6465 - accuracy: 0.6990 - val_loss: 0.9209 - val_accuracy: 0.5921
Epoch 8/35
21/21 [==============================] - 5s 239ms/step - loss: 0.6047 - accuracy: 0.7319 - val_loss: 0.7906 - val_accuracy: 0.6908
Epoch 9/35
21/21 [==============================] - 8s 365ms/step - loss: 0.4786 - accuracy: 0.7911 - val_loss: 1.0612 - val_accuracy: 0.5461
Epoch 10/35
21/21 [==============================] - 5s 237ms/step - loss: 0.5114 - accuracy: 0.8026 - val_loss: 0.8396 - val_accuracy: 0.6447
Epoch 11/35
21/21 [==============================] - 5s 225ms/step - loss: 0.4232 - accuracy: 0.8240 - val_loss: 1.0062 - val_accuracy: 0.6053
Epoch 12/35
21/21 [==============================] - 8s 389ms/step - loss: 0.4266 - accuracy: 0.8487 - val_loss: 0.8111 - val_accuracy: 0.6447
Epoch 13/35
21/21 [==============================] - 5s 236ms/step - loss: 0.3038 - accuracy: 0.9062 - val_loss: 0.8306 - val_accuracy: 0.6974
Epoch 14/35
21/21 [==============================] - 6s 267ms/step - loss: 0.2461 - accuracy: 0.9211 - val_loss: 0.8745 - val_accuracy: 0.6513
Epoch 15/35
21/21 [==============================] - 5s 240ms/step - loss: 0.2133 - accuracy: 0.9391 - val_loss: 0.9095 - val_accuracy: 0.6776
Epoch 16/35
21/21 [==============================] - 5s 228ms/step - loss: 0.1748 - accuracy: 0.9556 - val_loss: 0.8721 - val_accuracy: 0.6711
Epoch 17/35
21/21 [==============================] - 8s 352ms/step - loss: 0.1386 - accuracy: 0.9671 - val_loss: 1.0478 - val_accuracy: 0.6645
Epoch 18/35
21/21 [==============================] - 5s 230ms/step - loss: 0.1379 - accuracy: 0.9704 - val_loss: 0.9268 - val_accuracy: 0.6908
Epoch 19/35
21/21 [==============================] - 5s 229ms/step - loss: 0.1329 - accuracy: 0.9622 - val_loss: 0.9424 - val_accuracy: 0.7039
Epoch 20/35
21/21 [==============================] - 8s 376ms/step - loss: 0.0721 - accuracy: 0.9934 - val_loss: 1.0071 - val_accuracy: 0.6776
Epoch 21/35
21/21 [==============================] - 5s 238ms/step - loss: 0.0569 - accuracy: 0.9984 - val_loss: 1.0591 - val_accuracy: 0.6711
Epoch 22/35
21/21 [==============================] - 7s 310ms/step - loss: 0.0440 - accuracy: 1.0000 - val_loss: 1.0551 - val_accuracy: 0.6842
```

```
Epoch 23/35
21/21 [==============================] - 6s 286ms/step - loss: 0.0341 - accuracy: 0.9984 - val_loss: 1.1217 - val_accuracy: 0.6842
Epoch 24/35
21/21 [==============================] - 5s 236ms/step - loss: 0.0290 - accuracy: 1.0000 - val_loss: 1.1480 - val_accuracy: 0.7039
Epoch 25/35
21/21 [==============================] - 6s 286ms/step - loss: 0.0312 - accuracy: 0.9984 - val_loss: 1.1975 - val_accuracy: 0.6645
Epoch 26/35
21/21 [==============================] - 5s 227ms/step - loss: 0.0278 - accuracy: 1.0000 - val_loss: 1.2020 - val_accuracy: 0.6908
Epoch 27/35
21/21 [==============================] - 7s 324ms/step - loss: 0.0183 - accuracy: 1.0000 - val_loss: 1.2352 - val_accuracy: 0.6842
Epoch 28/35
21/21 [==============================] - 5s 228ms/step - loss: 0.0152 - accuracy: 1.0000 - val_loss: 1.2748 - val_accuracy: 0.6645
Epoch 29/35
21/21 [==============================] - 7s 311ms/step - loss: 0.0129 - accuracy: 1.0000 - val_loss: 1.2822 - val_accuracy: 0.6711
```

[ ]

```
fig, ax = plt.subplots(figsize=(10, 6))  # Adjust the figure size
ax.plot(fitting_performance_thrid.history['accuracy'], color='teal', label='Training Accuracy', linewidth=2)
ax.plot(fitting_performance_thrid.history['val_accuracy'], color='orange', label='Validation Accuracy', linewidth=2)
ax.set_title('Training and Validation Accuracy Comparison', fontsize=15)
ax.set_ylabel('Accuracy Values')
ax.set_xlabel('Epoch Count')
ax.grid(True, linestyle='--', alpha=0.7)
legend = ax.legend(loc='upper left', frameon=True, edgecolor='black')
legend.get_texts()[0].set_text('Training Accuracy')
legend.get_texts()[1].set_text('Validation Accuracy')
legend.get_frame().set_facecolor('lightgray')
plt.tight_layout()
plt.show()
```



On compare the 3 model, i saw thw model with kernel size 7 * 7 have best validation accuracy as shown in the graph on comparing with 3 * 3 and 5 * 5 kernel size. So, I chose the model with kernel size 7*7 is best for my dataset.

Double-click (or enter) to edit

✓ 0s completed at 11:14 AM