In [2]:
```
pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\programdata\anaconda3\lib\site-pac
kages (0.11.2)
Requirement already satisfied: matplotlib>=2.2 in c:\programdata\anaconda3\lib
\site-packages (from seaborn) (3.5.1)
Requirement already satisfied: numpy>=1.15 in c:\programdata\anaconda3\lib\site
-packages (from seaborn) (1.21.5)
Requirement already satisfied: scipy>=1.0 in c:\programdata\anaconda3\lib\site-
packages (from seaborn) (1.7.3)
Requirement already satisfied: pandas>=0.23 in c:\programdata\anaconda3\lib\sit
e-packages (from seaborn) (1.4.2)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\si
te-packages (from matplotlib>=2.2->seaborn) (9.0.1)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib
\site-packages (from matplotlib>=2.2->seaborn) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in c:\programdata\anaconda3\lib
\site-packages (from matplotlib>=2.2->seaborn) (3.0.4)
Requirement already satisfied: python-dateutil>=2.7 in c:\programdata\anaconda3
\lib\site-packages (from matplotlib>=2.2->seaborn) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3\li
b\site-packages (from matplotlib>=2.2->seaborn) (4.25.0)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\sit
e-packages (from matplotlib>=2.2->seaborn) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\li
b\site-packages (from matplotlib>=2.2->seaborn) (1.3.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\sit
e-packages (from pandas>=0.23->seaborn) (2021.3)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-pa
ckages (from python-dateutil>=2.7->matplotlib>=2.2->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
pip install pydotplus
```

```
Requirement already satisfied: pydotplus in c:\programdata\anaconda3\lib\site-p
ackages (2.0.2)Note: you may need to restart the kernel to use updated package
s.
Requirement already satisfied: pyparsing>=2.0.1 in c:\programdata\anaconda3\lib
\site-packages (from pydotplus) (3.0.4)
```

In [1]:
```
import matplotlib.pyplot as plt
from PIL import Image
import seaborn as sns
import numpy as np
import pandas as pd
import os
from tensorflow.keras.utils import to_categorical
from glob import glob


from sklearn.model_selection import train_test_split
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_repo
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras import layers
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten

from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt

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

In [2]:
```
df = pd.read_csv('skinCancer/abstract_metadata.csv')
df.head(10)
```

Out[2]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000673	ISIC_0029659	akiec	histo	70	female	face
1	HAM_0005282	ISIC_0025178	akiec	histo	65	male	lower extremity
2	HAM_0006002	ISIC_0029915	akiec	histo	50	female	face
3	HAM_0000549	ISIC_0029360	akiec	histo	70	male	upper extremity
4	HAM_0000549	ISIC_0026152	akiec	histo	70	male	upper extremity
5	HAM_0006875	ISIC_0026575	akiec	histo	80	male	face
6	HAM_0006875	ISIC_0030586	akiec	histo	80	male	face
7	HAM_0002644	ISIC_0029417	akiec	histo	80	female	neck
8	HAM_0005282	ISIC_0028730	akiec	histo	65	male	lower extremity
9	HAM_0006898	ISIC_0029041	akiec	histo	80	male	scalp

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
lesion_id       0
image_id        0
dx              0
dx_type         0
age             0
sex             0
localization    0
dtype: int64
```

In [4]:
```
df.count()
```

Out[4]:
```
lesion_id       700
image_id        700
dx              700
dx_type         700
age             700
sex             700
localization    700
dtype: int64
```

In [5]:
```
df['age'].fillna(int(df['age'].mean()),inplace=True)
df.isnull().sum()
```

Out[5]:
```
lesion_id       0
image_id        0
dx              0
dx_type         0
age             0
sex             0
localization    0
dtype: int64
```

In [6]:
```
lesion_type_dict = {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl': 'Benign keratosis-like lesions ',
    'bcc': 'Basal cell carcinoma',
    'akiec': 'Actinic keratoses',
    'vasc': 'Vascular lesions',
    'df': 'Dermatofibroma'
}
base_skin_dir = 'skinCancer'

imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
                     for x in glob(os.path.join(base_skin_dir, '*', '*.jpg'))}
```

In [7]:
```
df['path'] = df['image_id'].map(imageid_path_dict.get)
df['cell_type'] = df['dx'].map(lesion_type_dict.get)
df['cell_type_idx'] = pd.Categorical(df['cell_type']).codes
df.head()
```

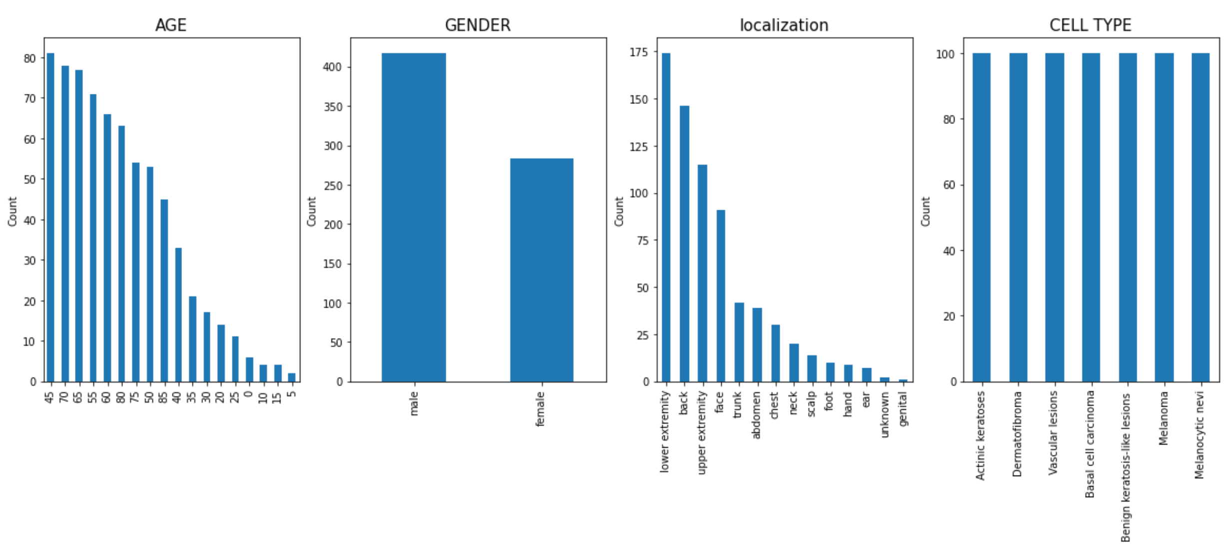
Out[7]:

	lesion_id	image_id	dx	dx_type	age	sex	localization	
0	HAM_0000673	ISIC_0029659	akiec	histo	70	female	face	skinCancer\HAM10000_imaç
1	HAM_0005282	ISIC_0025178	akiec	histo	65	male	lower extremity	skinCancer\HAM10000_imaç
2	HAM_0006002	ISIC_0029915	akiec	histo	50	female	face	skinCancer\HAM10000_imaç
3	HAM_0000549	ISIC_0029360	akiec	histo	70	male	upper extremity	skinCancer\HAM10000_imaç
4	HAM_0000549	ISIC_0026152	akiec	histo	70	male	upper extremity	skinCancer\HAM10000_imaç

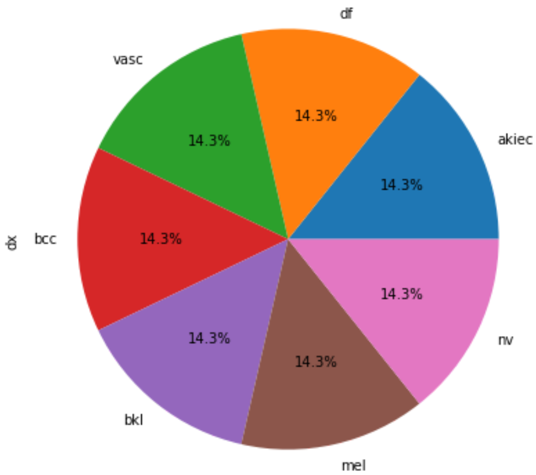
In [8]:
```
df['image'] = df['path'].map(lambda x: np.asarray(Image.open(x).resize((125,100))
```

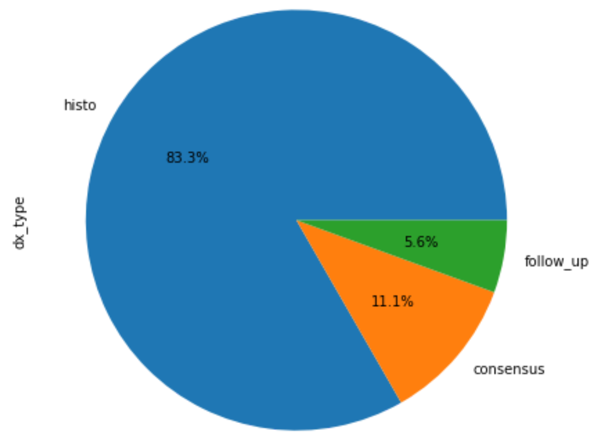
In [9]:
```
plt.figure(figsize=(20,10))
plt.subplots_adjust(left=0.125, bottom=1, right=0.9, top=2, hspace=0.2)
plt.subplot(2,4,1)
plt.title("AGE",fontsize=15)
plt.ylabel("Count")
df['age'].value_counts().plot.bar()

plt.subplot(2,4,2)
plt.title("GENDER",fontsize=15)
plt.ylabel("Count")
df['sex'].value_counts().plot.bar()

plt.subplot(2,4,3)
plt.title("localization",fontsize=15)
plt.ylabel("Count")
plt.xticks(rotation=45)
df['localization'].value_counts().plot.bar()

plt.subplot(2,4,4)
plt.title("CELL TYPE",fontsize=15)
plt.ylabel("Count")
df['cell_type'].value_counts().plot.bar()
```

Out[9]: <AxesSubplot:title={'center':'CELL TYPE'}, ylabel='Count'>



In [10]:
```
plt.figure(figsize=(15,10))
plt.subplot(1,2,1)
df['dx'].value_counts().plot.pie(autopct="%1.1f%%")
plt.subplot(1,2,2)
df['dx_type'].value_counts().plot.pie(autopct="%1.1f%%")
plt.show()
```





In [11]:
```
features=df.drop(columns=['cell_type_idx','image_id'],axis=1)
target=df['cell_type_idx']
features.head()
```

Out[11]:

	lesion_id	dx	dx_type	age	sex	localization	
0	HAM_0000673	akiec	histo	70	female	face	skinCancer\HAM10000_images_part_2\ISIC_
1	HAM_0005282	akiec	histo	65	male	lower extremity	skinCancer\HAM10000_images_part_1\ISIC_
2	HAM_0006002	akiec	histo	50	female	face	skinCancer\HAM10000_images_part_2\ISIC_
3	HAM_0000549	akiec	histo	70	male	upper extremity	skinCancer\HAM10000_images_part_2\ISIC_
4	HAM_0000549	akiec	histo	70	male	upper extremity	skinCancer\HAM10000_images_part_1\ISIC_

In [12]:
```
x_train_o, x_test_o, y_train_o, y_test_o = train_test_split(features, target, tes
tf.unique(x_train_o.cell_type.values)
```

Out[12]:
```
Unique(y=<tf.Tensor: shape=(7,), dtype=string, numpy=
array([b'Melanoma', b'Benign keratosis-like lesions ',
       b'Melanocytic nevi', b'Actinic keratoses', b'Dermatofibroma',
       b'Basal cell carcinoma', b'Vascular lesions'], dtype=object)>, idx=<tf.T
ensor: shape=(525,), dtype=int32, numpy=
array([0, 1, 0, 0, 0, 2, 3, 4, 0, 0, 1, 0, 3, 1, 2, 3, 5, 3, 2, 1, 4, 3,
       3, 1, 6, 0, 1, 4, 4, 0, 2, 6, 6, 0, 3, 5, 4, 2, 4, 6, 6, 2, 6, 3,
       1, 6, 2, 4, 3, 2, 5, 4, 1, 5, 1, 6, 2, 5, 3, 2, 5, 0, 0, 0, 4, 6,
       4, 4, 1, 4, 0, 2, 0, 0, 3, 5, 6, 6, 3, 5, 3, 3, 0, 0, 1, 1, 1, 6,
       5, 2, 3, 5, 3, 1, 4, 3, 0, 1, 5, 5, 0, 0, 6, 5, 3, 4, 0, 4, 4, 6,
       6, 4, 3, 4, 0, 5, 6, 1, 2, 5, 2, 3, 5, 4, 5, 0, 0, 2, 2, 3, 0, 6,
       4, 2, 3, 2, 5, 2, 1, 1, 6, 3, 0, 5, 0, 5, 2, 6, 6, 2, 6, 2, 2, 6,
       6, 1, 0, 2, 4, 2, 6, 1, 6, 4, 6, 2, 5, 1, 3, 2, 3, 3, 1, 5, 3, 3,
       4, 0, 2, 3, 5, 5, 1, 3, 3, 5, 4, 6, 0, 6, 3, 5, 6, 4, 0, 2, 2, 1,
       2, 6, 2, 4, 3, 1, 1, 5, 2, 0, 5, 6, 3, 4, 0, 5, 5, 3, 5, 2, 1, 3,
       6, 6, 2, 1, 6, 0, 1, 1, 4, 2, 6, 0, 6, 4, 1, 1, 5, 6, 6, 1, 0, 0,
       6, 4, 4, 4, 2, 1, 2, 2, 1, 6, 4, 0, 4, 3, 0, 1, 6, 5, 2, 5, 6, 0,
       1, 5, 6, 4, 3, 6, 6, 1, 2, 6, 5, 0, 4, 0, 2, 3, 3, 4, 6, 4, 6, 3,
       0, 4, 1, 0, 5, 0, 2, 5, 5, 2, 4, 2, 5, 2, 0, 6, 2, 0, 5, 1, 2, 5,
       1, 6, 2, 0, 3, 1, 4, 1, 2, 0, 4, 0, 6, 5, 6, 1, 1, 2, 0, 0, 5, 0,
       2, 3, 2, 1, 2, 0, 1, 2, 4, 5, 1, 4, 1, 2, 5, 2, 0, 4, 2, 2, 5, 4,
       5, 1, 4, 2, 1, 4, 1, 3, 0, 5, 6, 3, 4, 4, 6, 3, 3, 3, 4, 1, 4, 2,
       5, 5, 0, 6, 6, 0, 6, 4, 4, 2, 5, 1, 1, 4, 0, 5, 4, 3, 1, 1, 3, 5,
       6, 3, 6, 1, 0, 4, 5, 5, 2, 3, 3, 6, 0, 5, 2, 4, 6, 0, 1, 5, 3, 2,
       0, 2, 5, 0, 3, 1, 5, 4, 4, 6, 0, 3, 0, 6, 5, 1, 1, 5, 0, 3, 2, 5,
       0, 6, 0, 6, 6, 1, 5, 1, 1, 5, 4, 0, 5, 1, 0, 0, 5, 4, 3, 2, 6, 5,
       6, 4, 3, 3, 6, 5, 2, 4, 3, 3, 0, 4, 2, 3, 3, 2, 3, 5, 3, 0, 4, 2,
       6, 3, 6, 0, 3, 1, 2, 2, 4, 5, 2, 0, 6, 5, 3, 4, 4, 1, 2, 6, 4, 3,
       4, 6, 4, 6, 3, 1, 3, 1, 0, 4, 3, 0, 6, 3, 1, 1, 3, 1, 6])>)
```

In [13]:
```
x_train = np.asarray(x_train_o['image'].tolist())
x_test = np.asarray(x_test_o['image'].tolist())

x_train_mean = np.mean(x_train)
x_train_std = np.std(x_train)


x_test_mean = np.mean(x_test)
x_test_std = np.std(x_test)

x_train = (x_train - x_train_mean)/x_train_std
x_test = (x_test - x_test_mean)/x_test_std
```

In [14]:
```
# Perform one-hot encoding on the labels
y_train = to_categorical(y_train_o, num_classes = 7)
y_test = to_categorical(y_test_o, num_classes = 7)
y_test
```

Out[14]:
```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

In [15]:
```
y_test[1]
```

Out[15]:
```
array([0., 0., 0., 0., 0., 0., 1.], dtype=float32)
```

In [16]:
```
y_train
```

Out[16]:
```
array([[0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

In [17]:
```
y_test_o.value_counts()
```

Out[17]:
```
1    28
2    27
3    27
0    25
4    24
6    23
5    21
Name: cell_type_idx, dtype: int64
```

In [18]:
```
# x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train, t


# Reshape image in 3 dimensions (height = 100, width = 125 , canal = 3)
x_train = x_train.reshape(x_train_o.shape[0], *(100, 125, 3))
x_test = x_test.reshape(x_test_o.shape[0], *(100, 125, 3))
```

In [19]:
```
x_train = x_train.reshape(x_train.shape[0],125*100*3)
x_test = x_test.reshape(x_test.shape[0],125*100*3)
print(x_train.shape)
print(x_test.shape)
```

```
(525, 37500)
(175, 37500)
```

In [20]:
```
print(x_train)
```

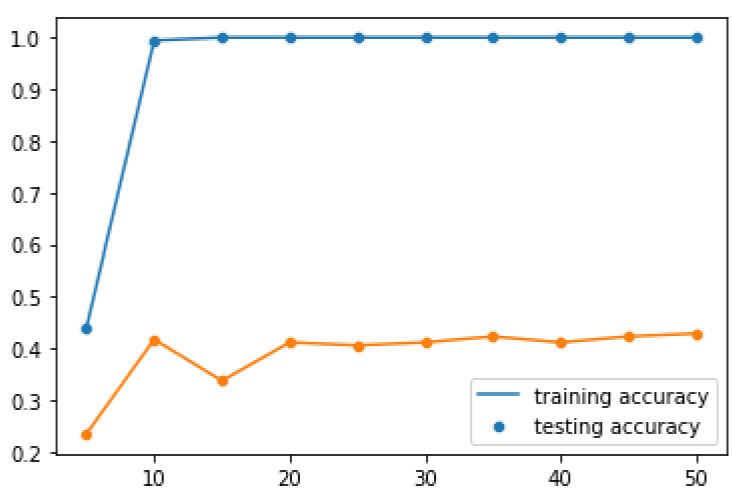
```
[[-0.09140732 -0.49162806 -0.46808566 ...  0.23818623 -0.02078013
   0.12047425]
 [ 0.07338946 -0.79767921 -0.09140732 ...  0.04984706 -0.44454326
   0.16755904]
 [-0.11494971 -0.44454326 -0.2091193  ...  0.02630467 -0.42100087
  -0.1855769 ]
 ...
 [ 1.22696689  0.23818623  0.70903416 ...  1.58010283 -0.1855769
   0.54423739]
 [-0.13849211 -0.70350962 -0.65642483 ... -0.32683128 -0.72705202
  -0.75059442]
 [ 1.60364523  0.61486458  0.70903416 ...  1.55656044  0.77966135
   0.77966135]]
```

Decision Tree

In [21]:
```
depth = range(5,51,5)
testing_accuracy = []
training_accuracy = []
score = 0

for i in depth:
    tree = DecisionTreeClassifier(max_depth = i, criterion = 'entropy')
    tree.fit(x_train, y_train)

    y_predict_train = tree.predict(x_train)
    training_accuracy.append(accuracy_score(y_train, y_predict_train))

    y_predict_test = tree.predict(x_test)
    acc_score = accuracy_score(y_test,y_predict_test)
    testing_accuracy.append(acc_score)

    print(i)

    if score < acc_score:
        score = acc_score
        best_depth = i

sns.lineplot(depth, training_accuracy)
sns.scatterplot(depth, training_accuracy)
sns.lineplot(depth, testing_accuracy)
sns.scatterplot(depth, testing_accuracy)
plt.legend(['training accuracy', 'testing accuracy'])
```

```
5
10
15
20
25
30
35
40
45
50
```

Out[21]:
```
<matplotlib.legend.Legend at 0x296633e2040>
```



In [22]:
```
print('This is the best depth for Decision Tree Classifier: ', best_depth, '\nAcc
result = confusion_matrix(y_test.argmax(axis=1), y_predict_test.argmax(axis=1),)
print("Confusion Matrix:","\n", result)
report = classification_report(y_test.argmax(axis=1), y_predict_test.argmax(axis=
print("Classification Report:" , "\n", report)
```

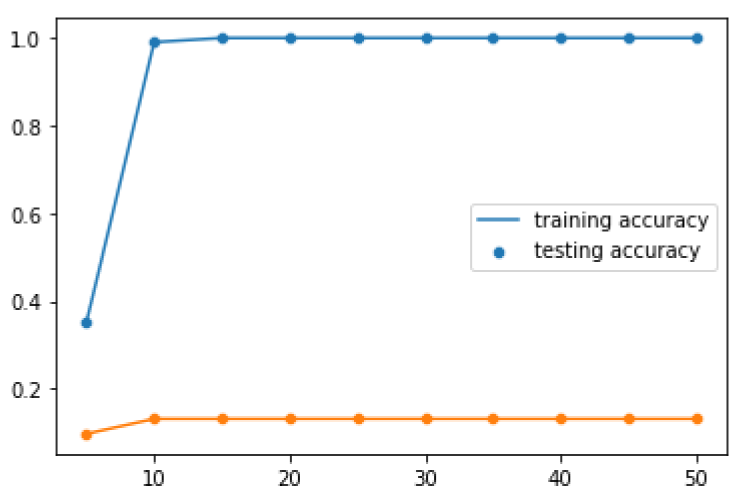
```
This is the best depth for Decision Tree Classifier:  50
Accuracy score is:  0.42857142857142855
Confusion Matrix:
 [[13  4  0  2  4  1  1]
 [ 5  7  2  8  3  1  2]
 [ 1  0 16  1  1  4  4]
 [ 6  2  2 12  0  2  3]
 [ 3  4  1  3 10  2  1]
 [ 5  0  4  1  1  8  2]
 [ 1  1  3  2  5  2  9]]
Classification Report:
               precision    recall  f1-score   support

           0       0.38      0.52      0.44        25
           1       0.39      0.25      0.30        28
           2       0.57      0.59      0.58        27
           3       0.41      0.44      0.43        27
           4       0.42      0.42      0.42        24
           5       0.40      0.38      0.39        21
           6       0.41      0.39      0.40        23

    accuracy                           0.43       175
   macro avg       0.43      0.43      0.42       175
weighted avg       0.43      0.43      0.42       175
```

Random Forest

In [23]:
```
#random Forest
depth = range(5,51,5)
testing_accuracy = []
training_accuracy = []
score = 0

for i in depth:
    tree = RandomForestClassifier(max_depth = i, criterion = 'gini', random_state
    tree.fit(x_train, y_train)

    y_predict_train = tree.predict(x_train)
    training_accuracy.append(accuracy_score(y_train, y_predict_train))

    y_predict_test = tree.predict(x_test)
    acc_score = accuracy_score(y_test,y_predict_test)
    testing_accuracy.append(acc_score)

    print(i)

    if score < acc_score:
        score = acc_score
        best_depth = i

sns.lineplot(depth, training_accuracy)
sns.scatterplot(depth, training_accuracy)
sns.lineplot(depth, testing_accuracy)
sns.scatterplot(depth, testing_accuracy)
plt.legend(['training accuracy', 'testing accuracy'])
```

```
5
10
15
20
25
30
35
40
45
50
```

Out[23]:
```
<matplotlib.legend.Legend at 0x296634e1af0>
```



In [24]:
```
print('This is the best depth for Decision Tree Classifier: ', best_depth, '\nAcc
result = confusion_matrix(y_test.argmax(axis=1), y_predict_test.argmax(axis=1),)
print("Confusion Matrix:","\n", result)
report = classification_report(y_test.argmax(axis=1), y_predict_test.argmax(axis=
print("Classification Report:" , "\n", report)
```

```
This is the best depth for Decision Tree Classifier:  10
Accuracy score is:  0.13142857142857142
Confusion Matrix:
 [[25  0  0  0  0  0  0]
 [25  2  0  0  1  0  0]
 [15  0  6  0  0  6  0]
 [26  0  0  1  0  0  0]
 [19  0  0  0  5  0  0]
 [15  0  2  0  0  4  0]
 [19  0  0  0  0  0  4]]
Classification Report:
               precision    recall  f1-score   support

           0       0.17      1.00      0.30        25
           1       1.00      0.07      0.13        28
           2       0.75      0.22      0.34        27
           3       1.00      0.04      0.07        27
           4       0.83      0.21      0.33        24
           5       0.40      0.19      0.26        21
           6       1.00      0.17      0.30        23

    accuracy                           0.27       175
   macro avg       0.74      0.27      0.25       175
weighted avg       0.75      0.27      0.24       175
```

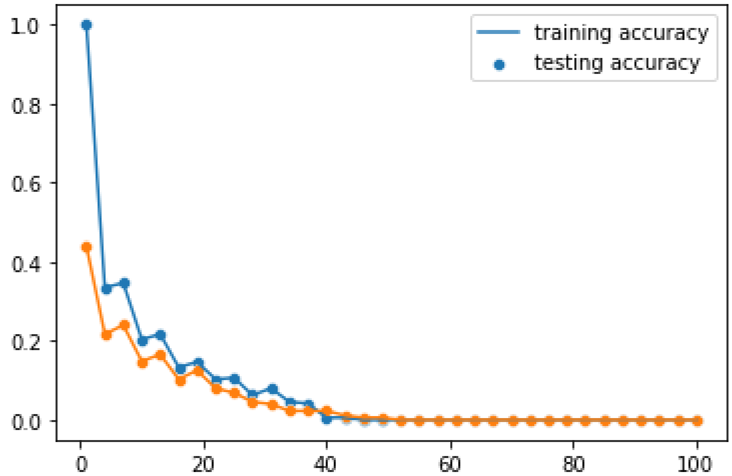
KNN

In [25]:
```
k = range(1,102,3)
testing_accuracy = []
training_accuracy = []
score = 0

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(x_train, y_train)

    y_predict_train = knn.predict(x_train)
    training_accuracy.append(accuracy_score(y_train, y_predict_train))

    y_predict_test = knn.predict(x_test)
    acc_score = accuracy_score(y_test,y_predict_test)
    testing_accuracy.append(acc_score)

    print(i)

    if score < acc_score:
        score = acc_score
        best_k = i


sns.lineplot(k, training_accuracy)
sns.scatterplot(k, training_accuracy)
sns.lineplot(k, testing_accuracy)
sns.scatterplot(k, testing_accuracy)
plt.legend(['training accuracy', 'testing accuracy'])
```

```
1
4
7
10
13
16
19
22
25
28
31
34
37
40
43
46
49
52
55
58
61
64
67
70
73
76
79
```

```
82
85
88
91
94
97
100
```

Out[25]: `<matplotlib.legend.Legend at 0x29663b0af40>`



In [26]:
```
print('This is the best depth for Decision Tree Classifier: ', best_depth, '\nAcc
result = confusion_matrix(y_test.argmax(axis=1), y_predict_test.argmax(axis=1),)
print("Confusion Matrix:","\n", result)
report = classification_report(y_test.argmax(axis=1), y_predict_test.argmax(axis=
print("Classification Report:" , "\n", report)
```

```
This is the best depth for Decision Tree Classifier:  10
Accuracy score is:  0.44
Confusion Matrix:
 [[25  0  0  0  0  0  0]
 [28  0  0  0  0  0  0]
 [27  0  0  0  0  0  0]
 [27  0  0  0  0  0  0]
 [24  0  0  0  0  0  0]
 [21  0  0  0  0  0  0]
 [23  0  0  0  0  0  0]]
Classification Report:
               precision    recall  f1-score   support

           0       0.14      1.00      0.25        25
           1       0.00      0.00      0.00        28
           2       0.00      0.00      0.00        27
           3       0.00      0.00      0.00        27
           4       0.00      0.00      0.00        24
           5       0.00      0.00      0.00        21
           6       0.00      0.00      0.00        23

    accuracy                           0.14       175
   macro avg       0.02      0.14      0.04       175
weighted avg       0.02      0.14      0.04       175
```

In [27]:
```
x_train.shape
```

Out[27]:
```
(525, 37500)
```

MLP

In [28]:
```
# define the keras model
model = Sequential()

model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu', i
model.add(Dense(units= 128, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units= 64, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units= 32, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units = 7, kernel_initializer = 'uniform', activation = 'softmax'
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 64)                2400064   
                                                                 
 dense_1 (Dense)             (None, 128)               8320      
                                                                 
 dense_2 (Dense)             (None, 64)                8256      
                                                                 
 dense_3 (Dense)             (None, 32)                2080      
                                                                 
 dense_4 (Dense)             (None, 7)                 231       
                                                                 
=================================================================
Total params: 2,418,951
Trainable params: 2,418,951
Non-trainable params: 0
_________________________________________________________________
```

In [29]:
```
# compile the keras model
model.compile(optimizer = "Adam", loss = 'categorical_crossentropy', metrics = ['

# fit the keras model on the dataset
history = model.fit(x_train, y_train, batch_size = 20, epochs = 25)
```

```
Epoch 1/25
27/27 [==============================] - 1s 17ms/step - loss: 1.8014 - accurac
y: 0.2419
Epoch 2/25
27/27 [==============================] - 0s 16ms/step - loss: 1.5498 - accurac
y: 0.2952
Epoch 3/25
27/27 [==============================] - 0s 17ms/step - loss: 1.4296 - accurac
y: 0.3695
Epoch 4/25
27/27 [==============================] - 0s 17ms/step - loss: 1.3027 - accurac
y: 0.4552
Epoch 5/25
27/27 [==============================] - 0s 16ms/step - loss: 1.2219 - accurac
y: 0.4990
Epoch 6/25
27/27 [==============================] - 0s 16ms/step - loss: 1.2073 - accurac
y: 0.5219
Epoch 7/25
27/27 [==============================] - 0s 16ms/step - loss: 1.1443 - accurac
y: 0.4743
Epoch 8/25
27/27 [==============================] - 0s 16ms/step - loss: 0.9984 - accurac
y: 0.5695
Epoch 9/25
27/27 [==============================] - 0s 16ms/step - loss: 0.9170 - accurac
y: 0.6095
Epoch 10/25
27/27 [==============================] - 0s 16ms/step - loss: 0.8727 - accurac
y: 0.6476
Epoch 11/25
27/27 [==============================] - 0s 16ms/step - loss: 0.7510 - accurac
y: 0.6800
Epoch 12/25
27/27 [==============================] - 0s 16ms/step - loss: 0.6952 - accurac
y: 0.7200
Epoch 13/25
27/27 [==============================] - 0s 16ms/step - loss: 0.6431 - accurac
y: 0.7333
Epoch 14/25
27/27 [==============================] - 0s 17ms/step - loss: 0.5594 - accurac
y: 0.7486
Epoch 15/25
27/27 [==============================] - 0s 16ms/step - loss: 0.6699 - accurac
y: 0.7562
Epoch 16/25
27/27 [==============================] - 0s 17ms/step - loss: 0.7145 - accurac
y: 0.7352
Epoch 17/25
27/27 [==============================] - 0s 16ms/step - loss: 0.5015 - accurac
y: 0.8057
```

```
Epoch 18/25
27/27 [==============================] - 0s 17ms/step - loss: 0.4875 - accurac
y: 0.8095
Epoch 19/25
27/27 [==============================] - 0s 17ms/step - loss: 0.3819 - accurac
y: 0.8610
Epoch 20/25
27/27 [==============================] - 0s 17ms/step - loss: 0.2616 - accurac
y: 0.8876
Epoch 21/25
27/27 [==============================] - 0s 17ms/step - loss: 0.3829 - accurac
y: 0.8571
Epoch 22/25
27/27 [==============================] - 0s 17ms/step - loss: 0.3570 - accurac
y: 0.8533
Epoch 23/25
27/27 [==============================] - 0s 17ms/step - loss: 0.3109 - accurac
y: 0.8971
Epoch 24/25
27/27 [==============================] - 0s 18ms/step - loss: 0.2291 - accurac
y: 0.9162
Epoch 25/25
27/27 [==============================] - 0s 17ms/step - loss: 0.2503 - accurac
y: 0.9048
```

In [30]:

```
accuracy = model.evaluate(x_test, y_test, verbose=1)[1]
print("Test: accuracy = ",accuracy*100,"%")
```

```
6/6 [==============================] - 0s 9ms/step - loss: 2.9224 - accuracy:
0.4229
Test: accuracy =  42.28571355342865 %
```

CNN

In [31]:

```
x_train = x_train.reshape(x_train.shape[0], 125,100,3)
x_test = x_test.reshape(x_test.shape[0], 125, 100, 3)
print(x_train.shape)
print(x_test.shape)
```

```
(525, 125, 100, 3)
(175, 125, 100, 3)
```

In [32]:
```
input_shape = (125, 100, 3)
num_classes = 7

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',padding = 'Same',input_
model.add(Conv2D(32,kernel_size=(3, 3), activation='relu',padding = 'Same'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(0.16))

model.add(Conv2D(64, (3, 3), activation='relu',padding = 'same'))
model.add(Conv2D(64, (3, 3), activation='relu',padding = 'Same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.20))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 125, 100, 32)      896       
                                                                 
 conv2d_1 (Conv2D)           (None, 125, 100, 32)      9248      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 62, 50, 32)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 62, 50, 32)        0         
                                                                 
 conv2d_2 (Conv2D)           (None, 62, 50, 64)        18496     
                                                                 
 conv2d_3 (Conv2D)           (None, 62, 50, 64)        36928     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 31, 25, 64)       0         
 2D)                                                             
                                                                 
 dropout_1 (Dropout)         (None, 31, 25, 64)        0         
                                                                 
 flatten (Flatten)           (None, 49600)             0         
                                                                 
 dense_5 (Dense)             (None, 256)               12697856  
                                                                 
 dense_6 (Dense)             (None, 128)               32896     
                                                                 
 dropout_2 (Dropout)         (None, 128)               0         
                                                                 
 dense_7 (Dense)             (None, 7)                 903       
                                                                 
=================================================================
Total params: 12,797,223
Trainable params: 12,797,223
```

```
Non-trainable params: 0
_________________________________________________________________
```

In [33]:

```
model.compile(optimizer= "Adam",
             loss='categorical_crossentropy',
             metrics=['accuracy'])

learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=4,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

In [34]:
```
#x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train, te


# Reshape image in 3 dimensions (height = 100, width = 125 , canal = 3)
x_train = x_train.reshape(x_train.shape[0], *(125, 100, 3))
x_test = x_test.reshape(x_test.shape[0], *(125,100, 3))
#x_validate = x_validate.reshape(x_validate.shape[0], *(100, 125, 3))

tf.config.run_functions_eagerly(True)


model.fit(x_train, y_train, batch_size = 100, epochs = 20)
```

```
Epoch 1/20
6/6 [==============================] - 36s 6s/step - loss: 2.7245 - accuracy:
0.1638
Epoch 2/20
6/6 [==============================] - 35s 6s/step - loss: 1.8154 - accuracy:
0.2229
Epoch 3/20
6/6 [==============================] - 35s 6s/step - loss: 1.6871 - accuracy:
0.2933
Epoch 4/20
6/6 [==============================] - 35s 6s/step - loss: 1.6260 - accuracy:
0.3010
Epoch 5/20
6/6 [==============================] - 35s 6s/step - loss: 1.5808 - accuracy:
0.2990
Epoch 6/20
6/6 [==============================] - 35s 6s/step - loss: 1.5184 - accuracy:
0.3714
Epoch 7/20
6/6 [==============================] - 35s 6s/step - loss: 1.4675 - accuracy:
0.4457
Epoch 8/20
6/6 [==============================] - 35s 6s/step - loss: 1.3887 - accuracy:
0.4514
Epoch 9/20
6/6 [==============================] - 35s 6s/step - loss: 1.3177 - accuracy:
0.4800
Epoch 10/20
6/6 [==============================] - 35s 6s/step - loss: 1.1809 - accuracy:
0.5295
Epoch 11/20
6/6 [==============================] - 35s 6s/step - loss: 1.1108 - accuracy:
0.5314
Epoch 12/20
6/6 [==============================] - 35s 6s/step - loss: 1.0377 - accuracy:
0.5657
Epoch 13/20
6/6 [==============================] - 35s 6s/step - loss: 0.9896 - accuracy:
0.6057
Epoch 14/20
6/6 [==============================] - 35s 6s/step - loss: 0.8993 - accuracy:
0.6495
Epoch 15/20
```

```
6/6 [==============================] - 35s 6s/step - loss: 0.8305 - accuracy:
0.6819
Epoch 16/20
6/6 [==============================] - 35s 6s/step - loss: 0.7601 - accuracy:
0.7010
Epoch 17/20
6/6 [==============================] - 35s 6s/step - loss: 0.7278 - accuracy:
0.7162
Epoch 18/20
6/6 [==============================] - 35s 6s/step - loss: 0.5842 - accuracy:
0.7714
Epoch 19/20
6/6 [==============================] - 35s 6s/step - loss: 0.5808 - accuracy:
0.7733
Epoch 20/20
6/6 [==============================] - 35s 6s/step - loss: 0.5257 - accuracy:
0.7752
```

Out[34]:

```
<keras.callbacks.History at 0x2966748f880>
```

In [35]:

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test Accuracy: ",test_acc*100,"%")
```

```
6/6 [==============================] - 4s 579ms/step - loss: 1.4076 - accuracy:
0.5200
Test Accuracy:  51.99999809265137 %
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: