In [2]:
```
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]:
```
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 [3]:
```
df = pd.read_csv('skinCancer/HAM10000_metadata.csv')
df.head(10)
```

Out[3]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear
5	HAM_0001466	ISIC_0027850	bkl	histo	75.0	male	ear
6	HAM_0002761	ISIC_0029176	bkl	histo	60.0	male	face
7	HAM_0002761	ISIC_0029068	bkl	histo	60.0	male	face
8	HAM_0005132	ISIC_0025837	bkl	histo	70.0	female	back
9	HAM_0005132	ISIC_0025209	bkl	histo	70.0	female	back

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
lesion_id        0
image_id         0
dx               0
dx_type          0
age             57
sex              0
localization     0
dtype: int64
```

In [5]:
```
df.count()
```

Out[5]:
```
lesion_id       10015
image_id        10015
dx              10015
dx_type         10015
age              9958
sex             10015
localization    10015
dtype: int64
```

In [6]:
```
df['age'].fillna(int(df['age'].mean()),inplace=True)
df.isnull().sum()
```

Out[6]:
```
lesion_id       0
image_id        0
dx              0
dx_type         0
age             0
sex             0
localization    0
dtype: int64
```

In [7]:
```
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 [8]:
```
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()
```

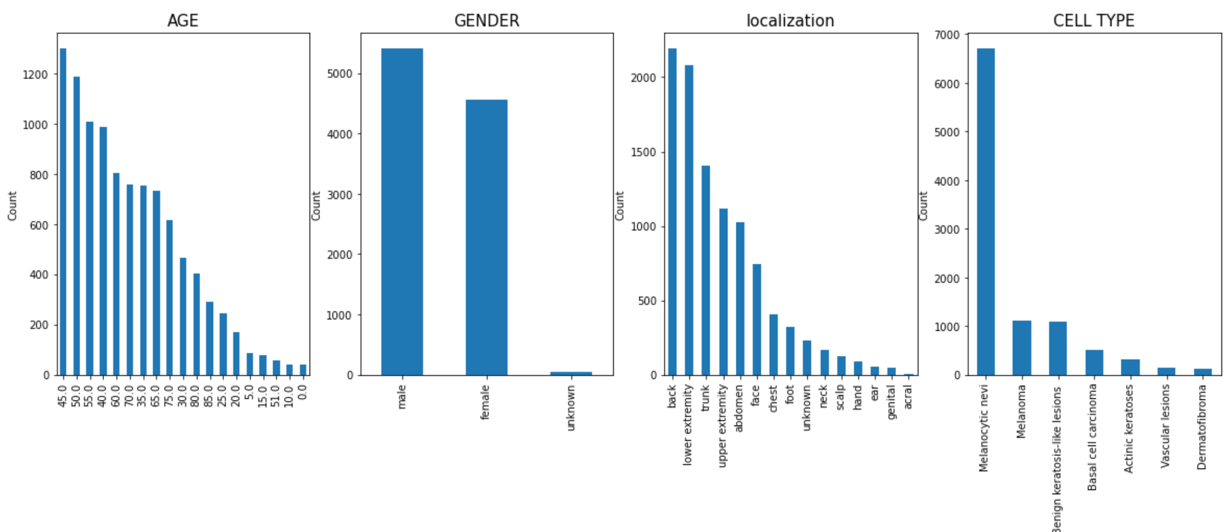
Out[8]:

	lesion_id	image_id	dx	dx_type	age	sex	localization	
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp	skinCancer\HAM10000_images_
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp	skinCancer\HAM10000_images_
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp	skinCancer\HAM10000_images_
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp	skinCancer\HAM10000_images_
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear	skinCancer\HAM10000_images_

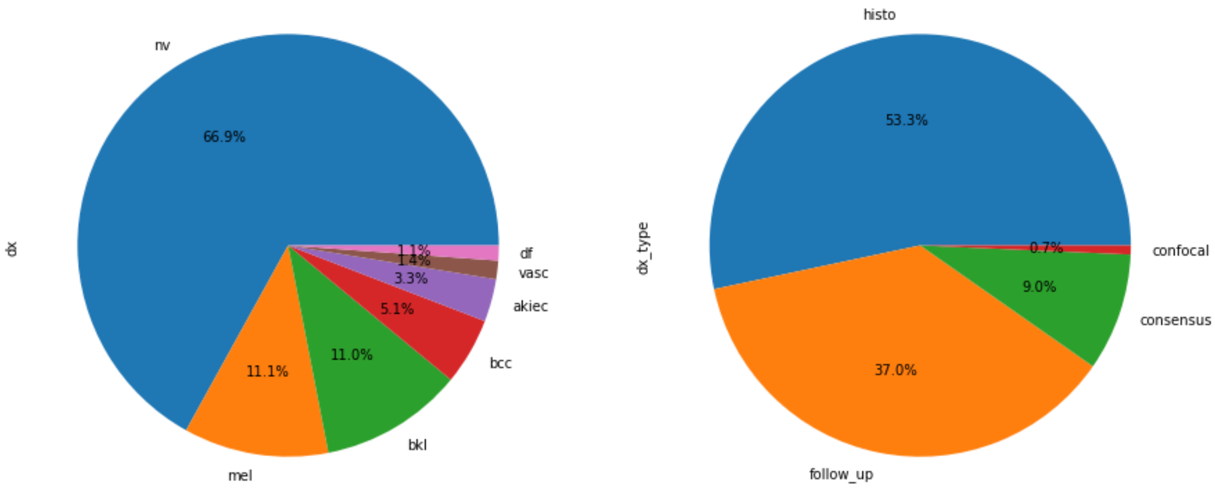
In [9]:
```
df['image'] = df['path'].map(lambda x: np.asarray(Image.open(x).resize((125,100)
```

In [10]:
```
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[10]: <AxesSubplot:title={'center':'CELL TYPE'}, ylabel='Count'>



In [11]:
```
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 [12]:
```
features=df.drop(columns=['cell_type_idx','image_id'],axis=1)
target=df['cell_type_idx']
features.head()
```

Out[12]:

	lesion_id	dx	dx_type	age	sex	localization	
0	HAM_0000118	bkl	histo	80.0	male	scalp	skinCancer\HAM10000_images_part_1\ISIC_002
1	HAM_0000118	bkl	histo	80.0	male	scalp	skinCancer\HAM10000_images_part_1\ISIC_002
2	HAM_0002730	bkl	histo	80.0	male	scalp	skinCancer\HAM10000_images_part_1\ISIC_002
3	HAM_0002730	bkl	histo	80.0	male	scalp	skinCancer\HAM10000_images_part_1\ISIC_002
4	HAM_0001466	bkl	histo	75.0	male	ear	skinCancer\HAM10000_images_part_2\ISIC_003

In [13]:
```
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[13]:
```
Unique(y=<tf.Tensor: shape=(7,), dtype=string, numpy=
array([b'Melanocytic nevi', b'Basal cell carcinoma', b'Melanoma',
       b'Vascular lesions', b'Benign keratosis-like lesions ',
       b'Actinic keratoses', b'Dermatofibroma'], dtype=object)>, idx=<tf.Tenso
r: shape=(7511,), dtype=int32, numpy=array([0, 1, 0, ..., 1, 0, 0])>)
```

In [14]:
```
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 [15]:
```
# 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[15]:
```
array([[0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.]], dtype=float32)
```

In [17]:
```
y_test[1]
```

Out[17]:
```
array([0., 0., 0., 0., 1., 0., 0.], dtype=float32)
```

In [18]:
```
y_train
```

Out[18]:
```
array([[0., 0., 0., ..., 1., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       ...,
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.]], dtype=float32)
```

In [19]:
```
y_test_o.value_counts()
```

Out[19]:
```
4    1693
2     277
5     274
1     123
0      68
6      39
3      30
Name: cell_type_idx, dtype: int64
```

In [16]:
```
# 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 [17]:
```
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)
```

```
(7511, 37500)
(2504, 37500)
```

In [22]:
```
print(x_train)
```

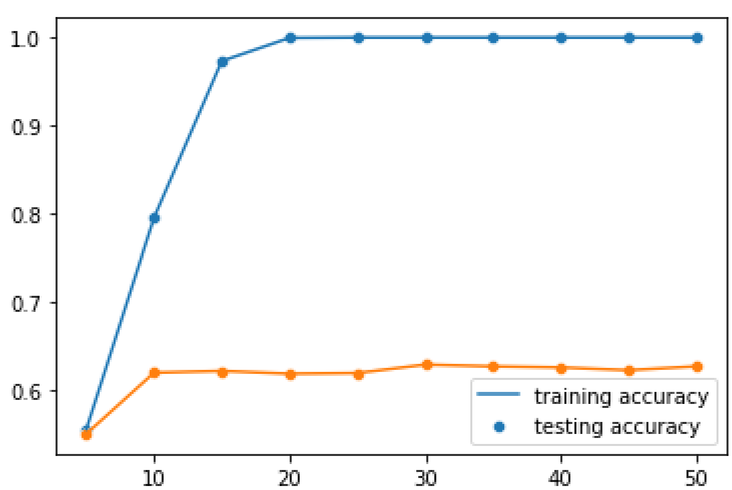
```
[[ 1.55231607e+00 -3.87567404e-01 -1.72024796e-01 ...  9.27242505e-01
  -5.60001490e-01 -6.03110012e-01]
 [ 3.23723203e-01 -4.09121664e-01 -1.72024796e-01 ...  3.66831724e-01
  -3.44458882e-01 -1.07362013e-01]
 [ 1.22900216e+00 -7.97098359e-01 -2.15133317e-01 ...  1.09967659e+00
  -3.87567404e-01 -2.15133317e-01]
 ...
 [-2.56454774e+00 -2.90941592e+00 -2.43522218e+00 ... -2.78009035e+00
  -2.99563296e+00 -2.65076479e+00]
 [ 4.53048767e-01 -1.07362013e-01 -8.58077525e-02 ...  3.88385985e-01
  -4.26992309e-02  4.09290704e-04]
 [ 1.63853311e+00 -3.01350360e-01 -2.15133317e-01 ...  1.09967659e+00
  -4.52230186e-01 -6.46218533e-01]]
```

Decision Tree

In [23]:
```
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[23]: <matplotlib.legend.Legend at 0x18880032ee0>



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:  30
Accuracy score is:  0.6285942492012779
Confusion Matrix:
 [[  17   12   13    1   20    5    0]
 [  17   26   30    5   33    6    6]
 [  12   19   90    9  100   45    2]
 [   3    4    6    1   10    5    1]
 [  12   35  116   10 1350  150   20]
 [  12   16   37    2  118   83    6]
 [   2    6    5    1   19    4    2]]
Classification Report:
               precision    recall  f1-score   support

           0       0.23      0.25      0.24        68
           1       0.22      0.21      0.22       123
           2       0.30      0.32      0.31       277
           3       0.03      0.03      0.03        30
           4       0.82      0.80      0.81      1693
           5       0.28      0.30      0.29       274
           6       0.05      0.05      0.05        39

    accuracy                           0.63      2504
   macro avg       0.28      0.28      0.28      2504
weighted avg       0.64      0.63      0.63      2504
```

Random Forest

In [ ]:
```
#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'])
```

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:  0.6202076677316294
Confusion Matrix:
 [[  63    1    0    0    4    0    0]
 [  99    7    2    0   15    0    0]
 [ 164    0   25    0   88    0    0]
 [  23    0    0    0    7    0    0]
 [ 221    0   10    0 1459    3    0]
 [ 149    0    3    0  114    8    0]
 [  21    0    0    0   18    0    0]]
Classification Report:
               precision    recall  f1-score   support

           0       0.09      0.93      0.16        68
           1       0.88      0.06      0.11       123
           2       0.62      0.09      0.16       277
           3       0.00      0.00      0.00        30
           4       0.86      0.86      0.86      1693
           5       0.73      0.03      0.06       274
           6       0.00      0.00      0.00        39

    accuracy                           0.62      2504
   macro avg       0.45      0.28      0.19      2504
weighted avg       0.77      0.62      0.61      2504
```

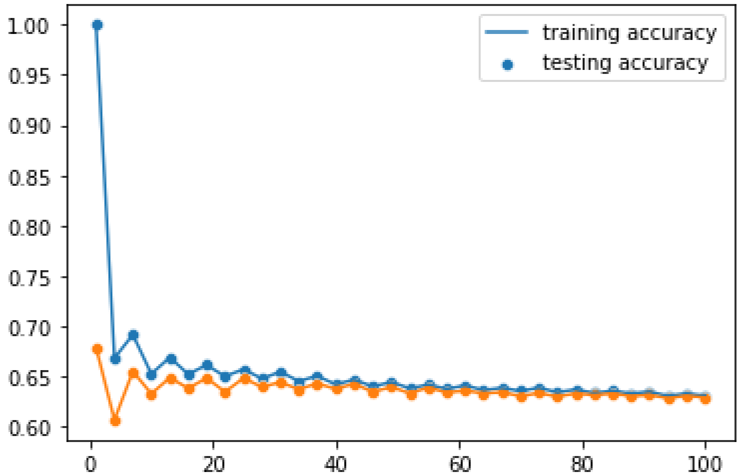
KNN

In [27]:
```
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[27]: <matplotlib.legend.Legend at 0x188f40ca340>



In [28]:
```
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:  0.6793130990415336
Confusion Matrix:
 [[  50    0    0    0   18    0    0]
 [  77    0    1    0   45    0    0]
 [ 131    0    6    0  140    0    0]
 [  13    0    1    0   16    0    0]
 [ 121    0    2    0 1570    0    0]
 [  82    0    5    0  187    0    0]
 [  13    0    0    0   26    0    0]]
Classification Report:
               precision    recall  f1-score   support

           0       0.10      0.74      0.18        68
           1       0.00      0.00      0.00       123
           2       0.40      0.02      0.04       277
           3       0.00      0.00      0.00        30
           4       0.78      0.93      0.85      1693
           5       0.00      0.00      0.00       274
           6       0.00      0.00      0.00        39

    accuracy                           0.65      2504
   macro avg       0.18      0.24      0.15      2504
weighted avg       0.58      0.65      0.58      2504
```

In [29]:
```
x_train.shape
```

Out[29]:
```
(7511, 37500)
```

MLP

In [30]:
```
# 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 [31]:
```
# 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
376/376 [==============================] - 9s 16ms/step - loss: 0.9923 - accura
cy: 0.6678
Epoch 2/25
376/376 [==============================] - 6s 16ms/step - loss: 0.8973 - accura
cy: 0.6865
Epoch 3/25
376/376 [==============================] - 6s 16ms/step - loss: 0.8596 - accura
cy: 0.7000
Epoch 4/25
376/376 [==============================] - 6s 16ms/step - loss: 0.8178 - accura
cy: 0.7092
Epoch 5/25
376/376 [==============================] - 6s 17ms/step - loss: 0.7929 - accura
cy: 0.7131
Epoch 6/25
376/376 [==============================] - 6s 17ms/step - loss: 0.7731 - accura
cy: 0.7256
Epoch 7/25
376/376 [==============================] - 6s 17ms/step - loss: 0.7331 - accura
cy: 0.7351
Epoch 8/25
376/376 [==============================] - 6s 17ms/step - loss: 0.7064 - accura
cy: 0.7383
Epoch 9/25
376/376 [==============================] - 6s 17ms/step - loss: 0.6761 - accura
cy: 0.7556
Epoch 10/25
376/376 [==============================] - 6s 16ms/step - loss: 0.6542 - accura
cy: 0.7585
Epoch 11/25
376/376 [==============================] - 6s 16ms/step - loss: 0.6337 - accura
cy: 0.7662
Epoch 12/25
376/376 [==============================] - 6s 17ms/step - loss: 0.6045 - accura
cy: 0.7759
Epoch 13/25
376/376 [==============================] - 6s 17ms/step - loss: 0.5875 - accura
cy: 0.7835
Epoch 14/25
376/376 [==============================] - 6s 17ms/step - loss: 0.5539 - accura
cy: 0.7988
Epoch 15/25
376/376 [==============================] - 6s 17ms/step - loss: 0.5272 - accura
cy: 0.8083
Epoch 16/25
376/376 [==============================] - 6s 17ms/step - loss: 0.5191 - accura
cy: 0.8083
Epoch 17/25
376/376 [==============================] - 6s 16ms/step - loss: 0.5057 - accura
cy: 0.8129
```

```
Epoch 18/25
376/376 [==============================] - 6s 16ms/step - loss: 0.4894 - accura
cy: 0.8223
Epoch 19/25
376/376 [==============================] - 6s 16ms/step - loss: 0.4619 - accura
cy: 0.8304
Epoch 20/25
376/376 [==============================] - 6s 16ms/step - loss: 0.4364 - accura
cy: 0.8386
Epoch 21/25
376/376 [==============================] - 6s 16ms/step - loss: 0.4161 - accura
cy: 0.8484
Epoch 22/25
376/376 [==============================] - 6s 16ms/step - loss: 0.4188 - accura
cy: 0.8466
Epoch 23/25
376/376 [==============================] - 6s 16ms/step - loss: 0.3984 - accura
cy: 0.8518
Epoch 24/25
376/376 [==============================] - 6s 16ms/step - loss: 0.3836 - accura
cy: 0.8662
Epoch 25/25
376/376 [==============================] - 6s 16ms/step - loss: 0.3669 - accura
cy: 0.8655
```

In [32]:

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

```
79/79 [==============================] - 1s 9ms/step - loss: 1.1677 - accuracy:
0.7017
Test: accuracy =  70.16773223876953 %
```

CNN

In [18]:

```
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)
```

```
(7511, 125, 100, 3)
(2504, 125, 100, 3)
```

In [19]:
```
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"
_________________________________________________________________
 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 (Dense)               (None, 256)               12697856  

 dense_1 (Dense)             (None, 128)               32896     

 dropout_2 (Dropout)         (None, 128)               0         

 dense_2 (Dense)             (None, 7)                 903       

=================================================================
Total params: 12,797,223
Trainable params: 12,797,223
```

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

In [20]:
```
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 [21]:
```
#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
76/76 [==============================] - 498s 7s/step - loss: 1.1379 - accurac
y: 0.6457
Epoch 2/20
76/76 [==============================] - 491s 6s/step - loss: 0.8668 - accurac
y: 0.6867
Epoch 3/20
76/76 [==============================] - 491s 6s/step - loss: 0.8072 - accurac
y: 0.7047
Epoch 4/20
76/76 [==============================] - 492s 6s/step - loss: 0.7610 - accurac
y: 0.7251
Epoch 5/20
76/76 [==============================] - 496s 7s/step - loss: 0.7106 - accurac
y: 0.7386
Epoch 6/20
76/76 [==============================] - 492s 6s/step - loss: 0.6676 - accurac
y: 0.7565
Epoch 7/20
76/76 [==============================] - 492s 6s/step - loss: 0.6182 - accurac
y: 0.7747
Epoch 8/20
76/76 [==============================] - 493s 6s/step - loss: 0.5940 - accurac
y: 0.7842
Epoch 9/20
76/76 [==============================] - 499s 7s/step - loss: 0.5232 - accurac
y: 0.8103
Epoch 10/20
76/76 [==============================] - 491s 6s/step - loss: 0.4530 - accurac
y: 0.8333
Epoch 11/20
76/76 [==============================] - 492s 6s/step - loss: 0.3860 - accurac
y: 0.8509
Epoch 12/20
76/76 [==============================] - 494s 6s/step - loss: 0.3036 - accurac
y: 0.8895
Epoch 13/20
76/76 [==============================] - 493s 6s/step - loss: 0.2649 - accurac
y: 0.9020
Epoch 14/20
76/76 [==============================] - 490s 6s/step - loss: 0.2175 - accurac
y: 0.9241
Epoch 15/20
```

```
76/76 [==============================] - 489s 6s/step - loss: 0.1590 - accurac
y: 0.9413
Epoch 16/20
76/76 [==============================] - 489s 6s/step - loss: 0.1416 - accurac
y: 0.9494
Epoch 17/20
76/76 [==============================] - 489s 6s/step - loss: 0.1038 - accurac
y: 0.9637
Epoch 18/20
76/76 [==============================] - 489s 6s/step - loss: 0.1187 - accurac
y: 0.9625
Epoch 19/20
76/76 [==============================] - 490s 6s/step - loss: 0.0746 - accurac
y: 0.9728
Epoch 20/20
76/76 [==============================] - 489s 6s/step - loss: 0.0760 - accurac
y: 0.9738
```

Out[21]:

```
<keras.callbacks.History at 0x201cb55cfd0>
```

In [22]:

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

```
79/79 [==============================] - 51s 642ms/step - loss: 1.5483 - accura
cy: 0.7420
Test Accuracy:  74.20127987861633 %
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: