In [ ]:
```
pip install tensorflow
```

In [ ]:
```
pip install sklearn
```

In [ ]:
```
pip install matplotlib
```

In [ ]:
```
pip install keras
```

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

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

In [1]:
```python
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]:
```python
df = pd.read_csv('skinCancer/HAM10000_metadata_Incereased.csv')
df.head(10)
```

Out[2]:

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

Out[3]:
```
lesion_id        0
image_id         0
dx               0
dx_type          0
age            119
sex              0
localization     0
dtype: int64
```

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

Out[4]:
```
lesion_id       49517
image_id        49517
dx              49517
dx_type         49517
age             49398
sex             49517
localization    49517
dtype: int64
```

In [5]:
```python
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]:
```python
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]:
```python
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_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 [8]:
```python
df['image'] = df['path'].map(lambda x: np.asarray(Image.open(x).resize((125,100))
```

In [9]:
```python
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()
```
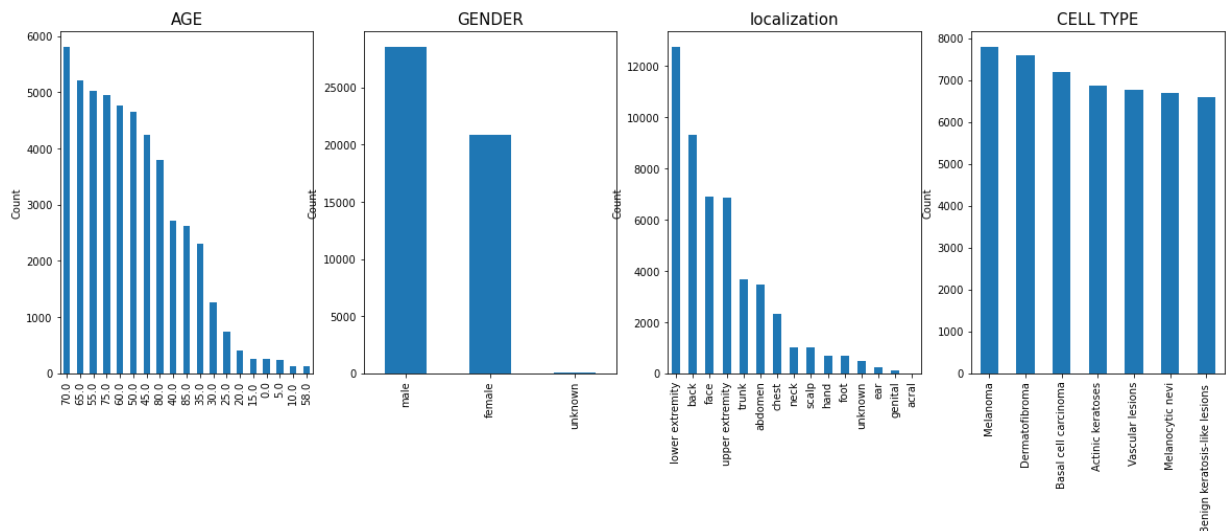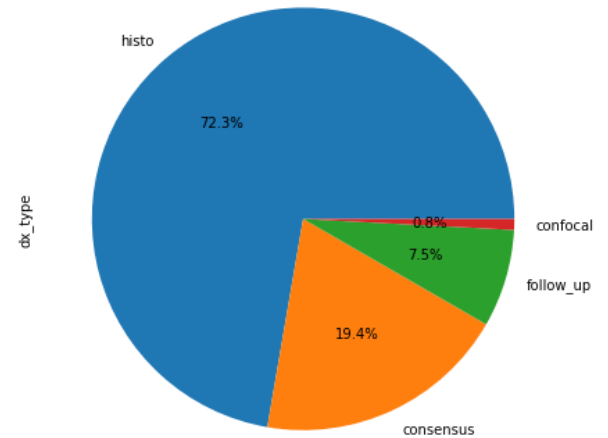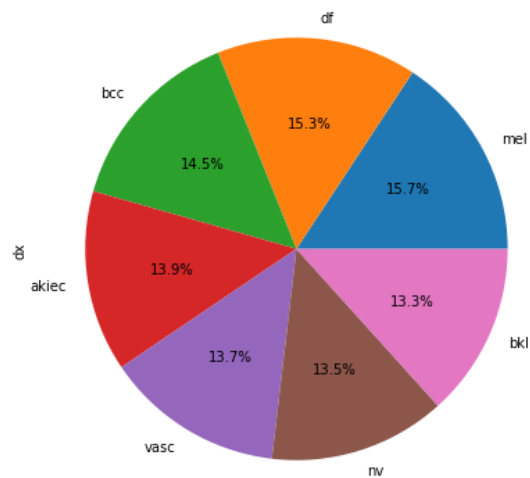
Out[9]: `<AxesSubplot:title={'center':'CELL TYPE'}, ylabel='Count'>`

In [10]:
```python
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]:
```python
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_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 [12]:
```python
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'Actinic keratoses', b'Basal cell carcinoma', b'Dermatofibroma',
       b'Melanoma', b'Benign keratosis-like lesions ',
       b'Melanocytic nevi', b'Vascular lesions'], dtype=object)>, idx=<tf.Tenso
r: shape=(37137,), dtype=int32, numpy=array([0, 1, 1, ..., 1, 1, 6])>)
```

In [13]:
```python
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]:
```python
# 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., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

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

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

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

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

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

Out[17]:
```
5    1924
3    1907
1    1750
0    1730
6    1697
2    1689
4    1683
Name: cell_type_idx, dtype: int64
```

In [18]:
```python
# 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]:
```python
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)
```
```
(37137, 37500)
(12380, 37500)
```

In [20]: `print(x_train)`
```
[[ 0.67592044 -0.3525907   0.07794885 ...  0.96294681  0.12578657
   0.31713748]
 [-0.04164547 -0.85488684 -1.54853389 ...  0.1736243  -0.6874548
  -1.14191321]
 [ 0.07794885 -1.33326412 -1.45285844 ...  0.60416385 -0.23299638
  -0.16123979]
 ...
 [-2.93582799 -3.31852981 -2.79231481 ... -3.17501663 -3.41420527
  -3.03150345]
 [ 0.53240726 -0.63961707 -0.71137366 ...  0.19754317 -1.02231889
  -1.14191321]
 [ 0.93902794 -0.30475297  0.19754317 ...  0.65200158 -0.63961707
  -0.23299638]]
```

Decision Tree

```
In [24]: depth = range(1,51,5)
         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'])
```
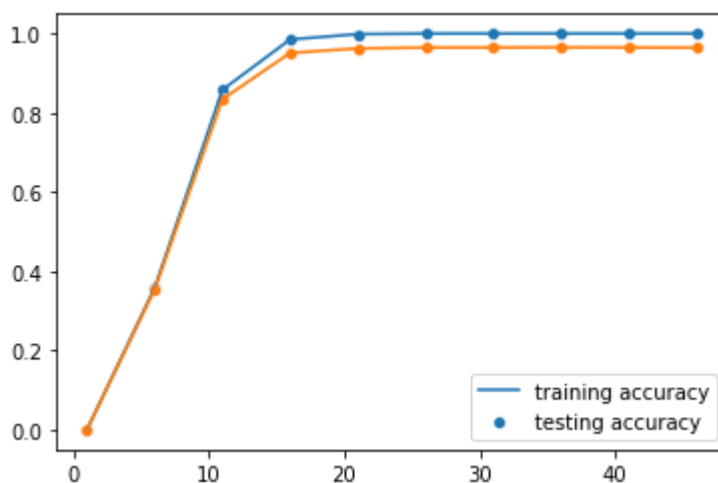
```
1
6
11
16
21
26
31
36
41
46
```

Out[24]: <matplotlib.legend.Legend at 0x14e31eaed90>

In [25]:
```python
print('This is the best depth for Decision Tree Classifier: ', best_depth, '\nAc
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:  36
Accuracy score is:  0.9651050080775444
Confusion Matrix:
 [[1702    0    7    0   21    0    0]
 [   0 1736    0    0   11    0    3]
 [   0    0 1689    0    0    0    0]
 [   0    0    0 1907    0    0    0]
 [  22   41  126    7 1284  176   27]
 [   0    0    2    0    0 1922    0]
 [   0    0    0    0    0    0 1697]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      1730
           1       0.98      0.99      0.98      1750
           2       0.93      1.00      0.96      1689
           3       1.00      1.00      1.00      1907
           4       0.98      0.76      0.86      1683
           5       0.92      1.00      0.96      1924
           6       0.98      1.00      0.99      1697

    accuracy                           0.96     12380
   macro avg       0.97      0.96      0.96     12380
weighted avg       0.97      0.96      0.96     12380
```

Random Forest

In [26]:
```python
#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'])
```
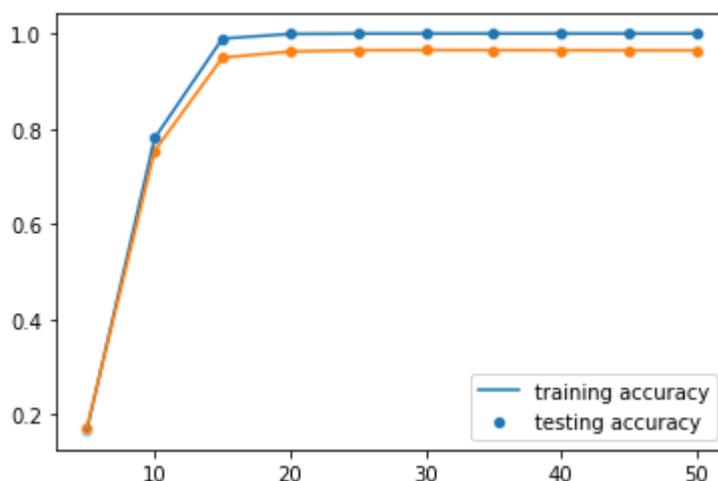
```
5
10
15
20
25
30
35
40
45
50
```

Out[26]: <matplotlib.legend.Legend at 0x14e31f2f880>

In [27]:
```python
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:  0.9652665589660743
Confusion Matrix:
 [[1730    0    0    0    0    0    0]
 [   0 1750    0    0    0    0    0]
 [   0    0 1689    0    0    0    0]
 [   0    0    0 1907    0    0    0]
 [ 398    3   15    0 1245   22    0]
 [   0    0    0    0    0 1924    0]
 [   0    0    0    0    0    0 1697]]
Classification Report:
              precision    recall  f1-score   support

           0       0.81      1.00      0.90      1730
           1       1.00      1.00      1.00      1750
           2       0.99      1.00      1.00      1689
           3       1.00      1.00      1.00      1907
           4       1.00      0.74      0.85      1683
           5       0.99      1.00      0.99      1924
           6       1.00      1.00      1.00      1697

    accuracy                           0.96     12380
   macro avg       0.97      0.96      0.96     12380
weighted avg       0.97      0.96      0.96     12380
```

KNN

In [21]:
```python
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'])
```
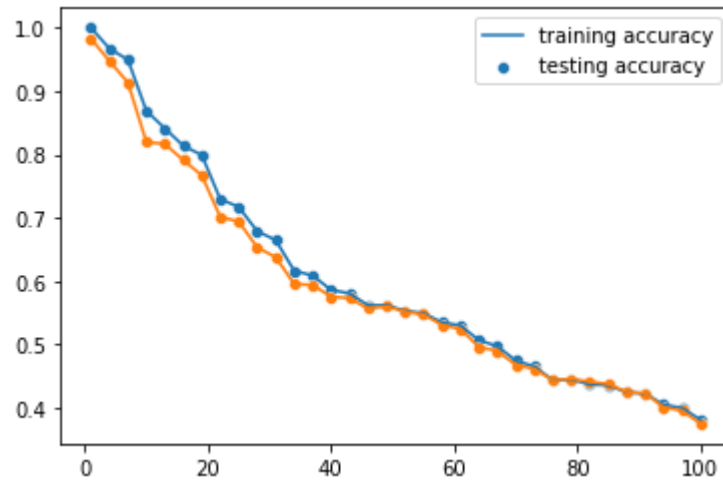
```
1
4
7
10
13
16
19
22
25
28
31
34
37
40
43
46
49
52
55
58
61
64
67
70
73
76
79
82
```

85
88
91
94
97
100

Out[21]: <matplotlib.legend.Legend at 0x12a87a9ac10>

In [28]:
```python
'''print('This is the best depth for Decision Tree Classifier: ', best_depth, '\n
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)
```

```
Confusion Matrix:
 [[1259  114   35  237    6    0   79]
 [ 862  476   45  279    0    0   88]
 [ 906   69  345  237   29   53   50]
 [ 543   27    0 1337    0    0    0]
 [ 627   25   34  158  686   58   95]
 [1248    6  137  105   45  289   94]
 [ 342   12    0   57   29    0 1257]]
Classification Report:
               precision    recall  f1-score   support

           0       0.22      0.73      0.33      1730
           1       0.65      0.27      0.38      1750
           2       0.58      0.20      0.30      1689
           3       0.55      0.70      0.62      1907
           4       0.86      0.41      0.55      1683
           5       0.72      0.15      0.25      1924
           6       0.76      0.74      0.75      1697

    accuracy                           0.46     12380
   macro avg       0.62      0.46      0.46     12380
weighted avg       0.62      0.46      0.45     12380
```

In [24]:
```python
x_train.shape
```

Out[24]: (37137, 37500)

MLP

In [25]:
```python
# 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 [26]:
```python
# 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
1857/1857 [==============================] - 26s 13ms/step - loss: 1.1413 - a
ccuracy: 0.5545
Epoch 2/25
1857/1857 [==============================] - 25s 14ms/step - loss: 0.6530 - a
ccuracy: 0.7451
Epoch 3/25
1857/1857 [==============================] - 25s 14ms/step - loss: 0.4905 - a
ccuracy: 0.8061
Epoch 4/25
1857/1857 [==============================] - 25s 14ms/step - loss: 0.4207 - a
ccuracy: 0.8389
Epoch 5/25
1857/1857 [==============================] - 23s 12ms/step - loss: 0.3594 - a
ccuracy: 0.8637
Epoch 6/25
1857/1857 [==============================] - 23s 12ms/step - loss: 0.3272 - a
ccuracy: 0.8782
Epoch 7/25
1857/1857 [==============================] - 25s 13ms/step - loss: 0.2870 - a
ccuracy: 0.8936
Epoch 8/25
1857/1857 [==============================] - 25s 13ms/step - loss: 0.2702 - a
ccuracy: 0.9006
Epoch 9/25
1857/1857 [==============================] - 25s 13ms/step - loss: 0.2527 - a
ccuracy: 0.9080
Epoch 10/25
1857/1857 [==============================] - 25s 13ms/step - loss: 0.2223 - a
ccuracy: 0.9213
Epoch 11/25
1857/1857 [==============================] - 25s 14ms/step - loss: 0.2130 - a
ccuracy: 0.9216
Epoch 12/25
1857/1857 [==============================] - 25s 13ms/step - loss: 0.2102 - a
ccuracy: 0.9269
Epoch 13/25
1857/1857 [==============================] - 25s 13ms/step - loss: 0.1950 - a
ccuracy: 0.9327
Epoch 14/25
1857/1857 [==============================] - 25s 14ms/step - loss: 0.1774 - a
ccuracy: 0.9366
Epoch 15/25
1857/1857 [==============================] - 26s 14ms/step - loss: 0.1805 - a
ccuracy: 0.9371
Epoch 16/25
1857/1857 [==============================] - 26s 14ms/step - loss: 0.1659 - a
ccuracy: 0.9420
Epoch 17/25
1857/1857 [==============================] - 26s 14ms/step - loss: 0.1665 - a
ccuracy: 0.9437
```

```
Epoch 18/25
1857/1857 [==============================] - 26s 14ms/step - loss: 0.1453 - a
ccuracy: 0.9501
Epoch 19/25
1857/1857 [==============================] - 27s 14ms/step - loss: 0.1458 - a
ccuracy: 0.9507
Epoch 20/25
1857/1857 [==============================] - 26s 14ms/step - loss: 0.1470 - a
ccuracy: 0.9504
Epoch 21/25
1857/1857 [==============================] - 26s 14ms/step - loss: 0.1352 - a
ccuracy: 0.9552
Epoch 22/25
1857/1857 [==============================] - 26s 14ms/step - loss: 0.1305 - a
ccuracy: 0.9572
Epoch 23/25
1857/1857 [==============================] - 26s 14ms/step - loss: 0.1333 - a
ccuracy: 0.9554
Epoch 24/25
1857/1857 [==============================] - 27s 14ms/step - loss: 0.1134 - a
ccuracy: 0.9616
Epoch 25/25
1857/1857 [==============================] - 26s 14ms/step - loss: 0.1235 - a
ccuracy: 0.9584
```

In [27]:
```python
accuracy = model.evaluate(x_test, y_test, verbose=1)[1]
print("Test: accuracy = ",accuracy*100,"%")
```

```
387/387 [==============================] - 5s 12ms/step - loss: 0.1793 - accura
cy: 0.9462
Test: accuracy =  94.62035298347473 %
```

CNN

In [ ]:
```python
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)
```

```python
In [ ]: 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()
```

```python
In [ ]: 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)
```

```python
In [ ]: #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], *(100, 125, 3))
        x_test = x_test.reshape(x_test.shape[0], *(100, 125, 3))
        #x_validate = x_validate.reshape(x_validate.shape[0], *(100, 125, 3))




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

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

```python
In [ ]:
```

```python
In [ ]:
```

```python
In [ ]:
```

```python
In [ ]:
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: