Importing Necessary Libraries

In [1]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import os
import random as rd
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.model_selection import train_test_split,GridSearchCV
import skimage.io
from skimage.transform import resize
```

Reading Pictures from Folder and storing Image Data, Flattened Data and Target in lists

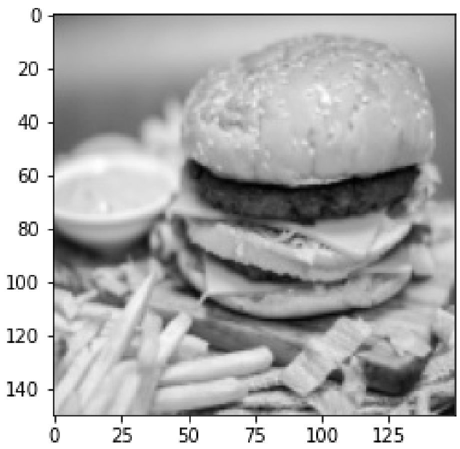
In [2]:

```
datadir="d:/Mini Project/"
food_list=["Burger","Pizza Slice"]
target=[]
flat_data=[]
image_data=[]

for i in food_list:
    path = os.path.join(datadir,i)
    for img in os.listdir(path):
        img_data = skimage.io.imread(os.path.join(path,img),as_gray=True)
        img_resized_data = resize(img_data,(150,150))
        image_data.append(img_resized_data)
        flat_data.append(np.ndarray.flatten(img_resized_data))
        target.append(i)
```

Displaying One Random Photo

In [3]:

```
plt.imshow(image_data[rd.randint(0,100)],cmap=plt.cm.gray)
plt.show()
```



Checking variables for training and changing to Numpy Arrray for faster calculation

In [4]:

```
print(len(flat_data),"\n",flat_data)
```

```
100
 [array([0.99943451, 0.99943451, 0.99943451, ..., 0.99943451, 0.99943451,
       0.99943451]), array([0.73770948, 0.74127664, 0.74537338, ..., 0.82608104, 0.820741
5 ,
       0.82495591]), array([0.57929625, 0.57964559, 0.58163728, ..., 0.81410886, 0.813006
23,
       0.81360249]), array([0.10386111, 0.10367134, 0.10164273, ..., 0.30235326, 0.301581
67,
       0.30316717]), array([0.        , 0.        , 0.        , ..., 0.88709865, 0.89049
443,
       0.8922065 ]), array([0.0581451 , 0.0581451 , 0.05818441, ..., 0.17721266, 0.165688
83,
       0.16445229]), array([0.96881657, 0.96426997, 0.871763  , ..., 0.57361284, 0.601924
47,
       0.71449144]), array([0.81083136, 0.8105101 , 0.8103634 , ..., 0.66255661, 0.812378
96,
       0.81622834]), array([1., 1., 1., ..., 1., 1., 1.]), array([0.01595412, 0.01595412,
0.01595412, ..., 0.06079967, 0.06079007,
       0.06079007]), array([0.16845068, 0.16843354, 0.17020453, ..., 0.90154757, 0.912222
28,
       0.90713936]), array([0.59275925, 0.64211733, 0.65969779, ..., 0.44653714, 0.491455
05,
       0.50002384]), array([0.99943451, 0.99943451, 0.99943451, ..., 0.99943451, 0.999434
51,
       0.99943451]), array([0.09003739, 0.087582  , 0.08738194, ..., 0.65186975, 0.660320
03,
       0.67224449]), array([0.05513084, 0.0545851 , 0.05447532, ..., 0.15169515, 0.152309
22,
       0.15177562]), array([0.85992948, 0.85685116, 0.85499854, ..., 0.2064018 , 0.208633
43,
       0.20602376]), array([1., 1., 1., ..., 1., 1., 1.]), array([1., 1., 1., ..., 1., 1.
, 1.]), array([0.08084512, 0.07914245, 0.08070697, ..., 0.08492843, 0.08765777,
       0.08689309]), array([0.55687353, 0.55706276, 0.56113421, ..., 0.8634483 , 0.864707
96,
       0.867721  ]), array([0.16696588, 0.16676663, 0.16639034, ..., 0.90402153, 0.893853
4 ,
       0.88240253]), array([0.2541429 , 0.26333858, 0.27245771, ..., 0.19815295, 0.209037
44,
       0.20603496]), array([1., 1., 1., ..., 1., 1., 1.]), array([0.21713705, 0.21900577,
0.21485407, ..., 0.2381709 , 0.23816738,
       0.23816715]), array([0.02379725, 0.02379701, 0.02339011, ..., 0.08532849, 0.072071
93,
       0.07037247]), array([1., 1., 1., ..., 1., 1., 1.]), array([0.49013188, 0.47242604,
0.45634283, ..., 0.00362472, 0.00267723,
       0.00299341]), array([0.77745705, 0.79297282, 0.79480935, ..., 0.5274951 , 0.576229
94,
       0.51291949]), array([1.        , 1.        , 1.        , ..., 0.89866566, 0.89673
742,
       0.89177325]), array([0.97647059, 0.97647059, 0.97647059, ..., 0.97647059, 0.976470
59,
       0.97647059]), array([0.07184014, 0.07184014, 0.07184014, ..., 0.15442535, 0.154133
3 ,
       0.15407629]), array([1., 1., 1., ..., 1., 1., 1.]), array([0.87447256, 0.87619822,
0.87669605, ..., 0.93554355, 0.9328721 ,
       0.92926549]), array([0.97227019, 0.97014892, 0.97039837, ..., 0.27246685, 0.266256
57,
       0.26109922]), array([0.66324587, 0.66003339, 0.66123132, ..., 0.74318293, 0.781272
57,
       0.83471787]), array([0.04595304, 0.04057242, 0.04633299, ..., 0.17615707, 0.171866
89,
       0.19148919]), array([0.37259945, 0.40048324, 0.41217813, ..., 0.54968154, 0.558279
84,
       0.55374047]), array([0.01607271, 0.0165169 , 0.01503905, ..., 0.23397882, 0.233978
```

```
82,
       0.23397882]), array([0.22439686, 0.2244028 , 0.22497996, ..., 0.2031997 , 0.203413
72,
       0.20340537]), array([1.        , 1.        , 0.99999879, ..., 0.57840485, 0.59680
129,
       0.60782209]), array([0.36212608, 0.36337465, 0.34838564, ..., 0.67908246, 0.706493
89,
       0.74523098]), array([0.31879732, 0.32294655, 0.33741269, ..., 0.05315886, 0.055991
5 ,
       0.05596092]), array([0.94955754, 0.93413983, 0.76652926, ..., 0.81748881, 0.816493
83,
       0.81371631]), array([0.86819962, 0.87587854, 0.88195209, ..., 0.5787487 , 0.727043
71,
       0.80517806]), array([1.        , 1.        , 1.        , ..., 0.4390193 , 0.60475
49 ,
       0.53309124]), array([1., 1., 1., ..., 1., 1., 1.]), array([0.85009587, 0.83730484,
0.83072178, ..., 0.98039216, 0.98039216,
       0.98039216]), array([0.446913  , 0.44880582, 0.44818566, ..., 0.06790214, 0.091814
49,
       0.08223076]), array([0.17584745, 0.17543884, 0.17519681, ..., 0.60780116, 0.604897
03,
       0.59823765]), array([0.11954506, 0.11951127, 0.12116471, ..., 0.7331194 , 0.739420
04,
       0.74024243]), array([0.58814471, 0.58814471, 0.58814471, ..., 0.58917108, 0.589174
53,
       0.58999283]), array([0.92310819, 0.91825558, 0.91903629, ..., 0.91351874, 0.915865
67,
       0.89671924]), array([0.93424207, 0.93793098, 0.93717202, ..., 0.20510431, 0.205104
31,
       0.20510431]), array([1., 1., 1., ..., 1., 1., 1.]), array([0.96862745, 0.96862745,
0.96862745, ..., 0.96862745, 0.96862745,
       0.96862745]), array([1., 1., 1., ..., 1., 1., 1.]), array([0.49466434, 0.49666174,
0.49008782, ..., 0.51098389, 0.4991383 ,
       0.53116061]), array([0.40791226, 0.42166209, 0.42536601, ..., 0.39441216, 0.394412
16,
       0.39441216]), array([0.96595958, 0.96553596, 0.96553361, ..., 0.96553361, 0.965534
84,
       0.96575688]), array([0.4418708 , 0.41875255, 0.40855313, ..., 0.48824712, 0.497096
31,
       0.50711878]), array([0.12379508, 0.31588887, 0.40741309, ..., 0.85266106, 0.850521
73,
       0.85045466]), array([0.96614446, 0.96572084, 0.96571849, ..., 0.96571849, 0.965709
83,
       0.96560781]), array([0.96862745, 0.96862745, 0.96862745, ..., 0.96862745, 0.968627
45,
       0.96862745]), array([0.01503903, 0.01655877, 0.01462774, ..., 0.25808732, 0.404116
51,
       0.36860849]), array([1., 1., 1., ..., 1., 1., 1.]), array([1., 1., 1., ..., 1., 1.
, 1.]), array([1., 1., 1., ..., 1., 1., 1.]), array([1.        , 1.        , 1.        ,
..., 0.99934519, 0.99761782,
       0.98987501]), array([0.65174372, 0.45267937, 0.61627231, ..., 0.44676278, 0.522141
21,
       0.29628594]), array([0.99685155, 0.99648241, 0.99615729, ..., 0.98360016, 0.933864
35,
       0.92981585]), array([0.99911256, 0.99893636, 0.99871882, ..., 0.99850679, 0.998824
01,
       0.99928855]), array([0.96570248, 0.96527887, 0.96527652, ..., 0.96527652, 0.965272
66,
       0.96540007]), array([0.96862745, 0.96862745, 0.96862745, ..., 0.96862745, 0.968627
45,
       0.96862745]), array([1.        , 1.        , 1.        , ..., 0.20510431, 0.20510
431,
       0.20510431]), array([0.96107952, 0.96117788, 0.96362628, ..., 0.45148638, 0.444903
72,
       0.44433206]), array([0.99607843, 0.99607843, 0.99607843, ..., 0.99607843, 0.996078
43,
       0.99607843]), array([1.        , 1.        , 1.        , ..., 0.18130784, 0.18130
784,
       0.18130784]), array([0.96862745, 0.96862745, 0.96862745, ..., 0.96862745, 0.968627
45,
       0.96862745]), array([0.96862745, 0.96862745, 0.96862745, ..., 0.96862745, 0.968627
```

```
        0.96862745]), array([0.        , 0.        , 0.        , ..., 0.21397905, 0.01655
087,
        0.01298874]), array([1., 1., 1., ..., 1., 1., 1.]), array([1., 1., 1., ..., 1., 1.
, 1.]), array([0.37094745, 0.37316244, 0.36141877, ..., 0.56591605, 0.56203297,
        0.55506872]), array([0.03336204, 0.0278908 , 0.03044917, ..., 0.3424214 , 0.352771
57,
        0.34758634]), array([0.08994679, 0.08964179, 0.09797587, ..., 1.        , 1.
,
        1.        ]), array([0.4757767 , 0.64618765, 0.78820873, ..., 0.15719895, 0.170309
02,
        0.18369769]), array([1., 1., 1., ..., 1., 1., 1.]), array([1., 1., 1., ..., 1., 1.
, 1.]), array([1., 1., 1., ..., 1., 1., 1.]), array([0.91963263, 0.91999537, 0.91962347,
..., 0.39441216, 0.39441216,
        0.39441216]), array([1., 1., 1., ..., 1., 1., 1.]), array([1.        , 1.        ,
1.        , ..., 0.07484431, 0.07484431,
        0.07484431]), array([1., 1., 1., ..., 1., 1., 1.]), array([0.99775294, 0.99775294,
0.99775294, ..., 0.99943456, 0.99943451,
        0.99943451]), array([0.90836964, 0.90968711, 0.91049463, ..., 0.910947  , 0.905499
91,
        0.89246574]), array([0.06889291, 0.04462184, 0.03781061, ..., 0.28086506, 0.385036
03,
        0.16291819]), array([1., 1., 1., ..., 1., 1., 1.]), array([1., 1., 1., ..., 1., 1.
, 1.]), array([0.99999853, 0.9999999 , 1.        , ..., 0.20510431, 0.20510431,
        0.20510431]), array([1.        , 0.99912772, 0.95098039, ..., 0.95098039, 0.999127
72,
        1.        ])]
```

In [5]:

```python
print(len(target),"\n",target)
```

```
100
 ['Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger
', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burge
r', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burg
er', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Bur
ger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Bu
rger', 'Burger', 'Burger', 'Burger', 'Burger', 'Burger', 'Pizza Slice', 'Pizza Slice', 'P
izza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', '
Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice',
'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice',
'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice',
'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice',
'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice',
'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice',
'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice', 'Pizza Slice']
```

In [6]:

```python
flat_data = np.array(flat_data)
target = np.array(target)
```

In [7]:

```python
print(type(flat_data),type(target))
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

Encoding Target Data Values

In [8]:

```python
encoded_target=[]
ec=0
for i in range(0,len(target)-1):
    if(target[i]==target[i+1]):
        encoded_target.append(ec)
        continue
    else:
```

```
        encoded_target.append(ec)
        ec+=1
encoded_target.append(ec)
```

In [9]:

```
print(len(target),len(encoded_target),sep=",")
print("\n",encoded_target)
```

```
100,100

 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

Displaying all data for Model Training in a Dataframe

In [10]:

```
Model_Data=pd.DataFrame(flat_data)
Model_Data["Food"]=encoded_target
Model_Data
```

Out[10]:

	0	1	2	3	4	5	6	7	8	9	...	22491	2249
0	0.999435	0.999435	0.999435	0.999435	0.999435	0.999435	0.999435	0.999435	0.999435	0.999435	...	0.999435	0.99943
1	0.737709	0.741277	0.745373	0.750857	0.749768	0.748279	0.751153	0.754497	0.756580	0.740998	...	0.841512	0.83717
2	0.579296	0.579646	0.581637	0.576414	0.569804	0.562412	0.561701	0.568702	0.571939	0.571194	...	0.384741	0.37547
3	0.103861	0.103671	0.101643	0.101325	0.098693	0.095833	0.088152	0.085502	0.079144	0.077278	...	0.285230	0.28730
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.879776	0.87968
...	...	...	...	...	...	...	...	...	...	...	...	...	..
95	0.068893	0.044622	0.037811	0.055967	0.099826	0.089831	0.069332	0.077169	0.091657	0.101930	...	0.312210	0.33194
96	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.00000
97	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.00000
98	0.999999	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	0.205104	0.20510
99	1.000000	0.999128	0.950980	0.902850	0.913547	0.988414	0.999110	0.950980	0.902850	0.913547	...	0.902850	0.95098

100 rows × 22501 columns

Splitting Data for Training and Testing

In [11]:

```
x_train,x_test,y_train,y_test=train_test_split(flat_data,encoded_target,test_size=0.25,r
andom_state=42)
```

Logistic Regression

In [12]:

```
lgr_param_grid={"solver":['newton-cg','liblinear'],"C":[1,3,5,7,9]}
lgr_model=LogisticRegression()
lgr_cv=GridSearchCV(lgr_model,lgr_param_grid)
lgr_cv.fit(x_train,y_train)
```

Out[12]:

```
GridSearchCV(estimator=LogisticRegression(),
             param_grid={'C': [1, 3, 5, 7, 9],
                         'solver': ['newton-cg', 'liblinear']})
```

In [13]:

```
lgr_cv.best_params_
```

Out[13]:

```
{'C': 5, 'solver': 'newton-cg'}
```

In [14]:

```
lgr_predict=lgr_cv.predict(x_test)
```

In [15]:

```
lgr_accuracy=accuracy_score(y_test,lgr_predict)*100
lgr_cm=confusion_matrix(y_test,lgr_predict)
lgr_cr=classification_report(y_test,lgr_predict)
```

Support Vector Machine

In [16]:

```
svm_model=SVC()
svm_param_grid={"kernel": ['linear', 'poly', 'rbf'],"C":[1,3,5,7,9]}
svm_cv=GridSearchCV(svm_model,svm_param_grid)
svm_cv.fit(x_train,y_train)
```

Out[16]:

```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 3, 5, 7, 9],
                         'kernel': ['linear', 'poly', 'rbf']})
```

In [17]:

```
svm_cv.best_params_
```

Out[17]:

```
{'C': 1, 'kernel': 'linear'}
```

In [18]:

```
svm_predict=svm_cv.predict(x_test)
```

In [19]:

```
svm_accuracy=accuracy_score(y_test,svm_predict)*100
svm_cm=confusion_matrix(y_test,svm_predict)
svm_cr=classification_report(y_test,svm_predict)
```

Decision Tree

In [20]:

```
dect=DecisionTreeClassifier()
dt_param_grid={'criterion':["gini","entropy"],'splitter':["best","random"]}
dtree=GridSearchCV(dect,dt_param_grid)
dtree.fit(x_train,y_train)
dtree_pred=dtree.predict(x_test)
```

In [21]:

```
dtree.best_params_
```

```
dtree.best_params_
```

Out[21]:

```
{'criterion': 'gini', 'splitter': 'best'}
```

In [22]:

```
dt_accuracy=accuracy_score(y_test,dtree_pred)*100
dt_cm=confusion_matrix(y_test,dtree_pred)
dt_cr=classification_report(y_test,dtree_pred)
```

Random Forest

In [23]:

```
rfc=RandomForestClassifier()
rf_param_grid={"n_estimators": [i for i in range(1,50)],"criterion":["gini","entropy"]}
rf=GridSearchCV(rfc,rf_param_grid)
rf.fit(x_train,y_train)
rfpred=rf.predict(x_test)
```

In [24]:

```
rf.best_params_
```

Out[24]:

```
{'criterion': 'entropy', 'n_estimators': 27}
```

In [25]:

```
rf_accuracy=accuracy_score(y_test,rfpred)*100
rf_cm=confusion_matrix(y_test,rfpred)
rf_cr=classification_report(y_test,rfpred)
```

K Neighbors

In [26]:

```
kn=KNeighborsClassifier()
kn_param_grid={"n_neighbors" : [i for i in range(1,10)],"weights":["uniform","distance"]
}
kn_model=GridSearchCV(kn,kn_param_grid)
kn_model.fit(x_train,y_train)
kn_pred=kn_model.predict(x_test)
```

In [27]:

```
kn_model.best_params_
```

Out[27]:

```
{'n_neighbors': 3, 'weights': 'uniform'}
```

In [28]:

```
kn_accuracy=accuracy_score(y_test,kn_pred)*100
kn_cm=confusion_matrix(y_test,kn_pred)
kn_cr=classification_report(y_test,kn_pred)
```

Displaying Scores of Various Models

In [29]:

```
Model_Scores=pd.DataFrame({"K Neighbours":[kn_accuracy],
                           "Logistic Regression" : [lgr_accuracy],
```

```
                            "Support Vector Machine": [svm_accuracy],
                            "Decision Tree":[dt_accuracy],
                            "Random Forest":[rf_accuracy]},index=["Accuracy Score"])
```

In [30]:

```
Model_Scores
```

Out[30]:

	K Neighbours	Logistic Regression	Support Vector Machine	Decision Tree	Random Forest
Accuracy Score	84.0	80.0	72.0	72.0	84.0

Classification Report and Confusion Matrix for Logistic Regression

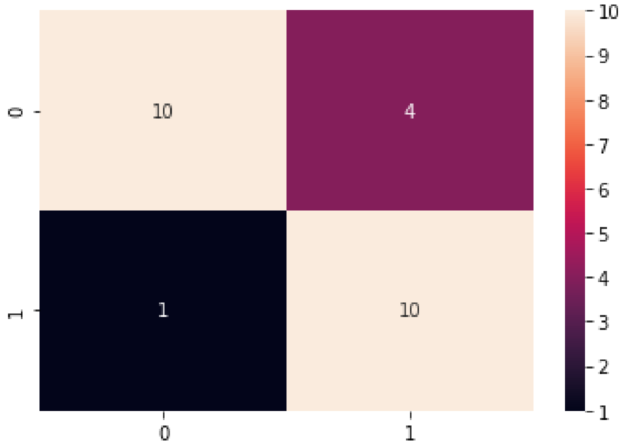
In [31]:

```
print(lgr_cr)
print("\n\tLOGISTIC REGRESSION")
sns.heatmap(lgr_cm,annot=True)
plt.plot()
```

```
              precision    recall  f1-score   support

           0       0.91      0.71      0.80        14
           1       0.71      0.91      0.80        11

    accuracy                           0.80        25
   macro avg       0.81      0.81      0.80        25
weighted avg       0.82      0.80      0.80        25


 LOGISTIC REGRESSION
```

Out[31]:

```
[]
```



Classification Report and Confusion Matrix for Support Vector Machine

In [32]:

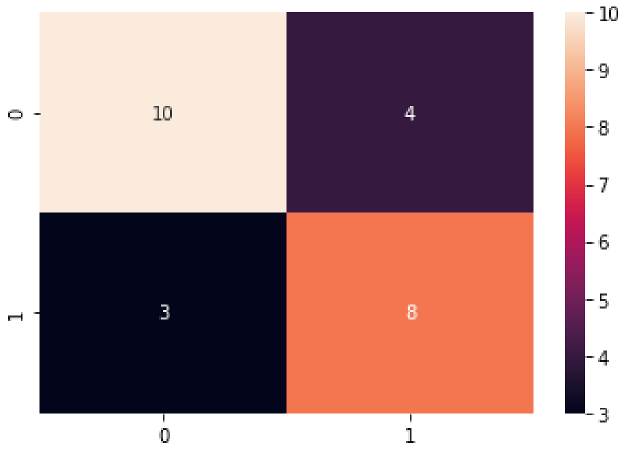
```
print(svm_cr)
print("\n\tSUPPORT VECTOR MACHINE")
sns.heatmap(svm_cm,annot=True)
plt.plot()
```

```
              precision    recall  f1-score   support
```

```
           0       0.77      0.71      0.74        14
           1       0.67      0.73      0.70        11

    accuracy                           0.72        25
   macro avg       0.72      0.72      0.72        25
weighted avg       0.72      0.72      0.72        25


 SUPPORT VECTOR MACHINE
```

Out[32]:

```
[]
```



Classification Report and Confusion Matrix for K Neighbours

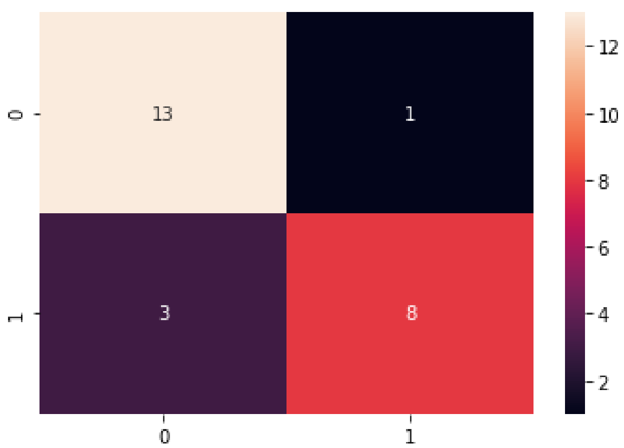
In [33]:

```python
print(kn_cr)
print("\n\tK Neighbours")
sns.heatmap(kn_cm,annot=True)
plt.plot()
```

```
              precision    recall  f1-score   support

           0       0.81      0.93      0.87        14
           1       0.89      0.73      0.80        11

    accuracy                           0.84        25
   macro avg       0.85      0.83      0.83        25
weighted avg       0.85      0.84      0.84        25


 K Neighbours
```

Out[33]:

```
[]
```



Classification Report and Confusion Matrix for Decision Tree

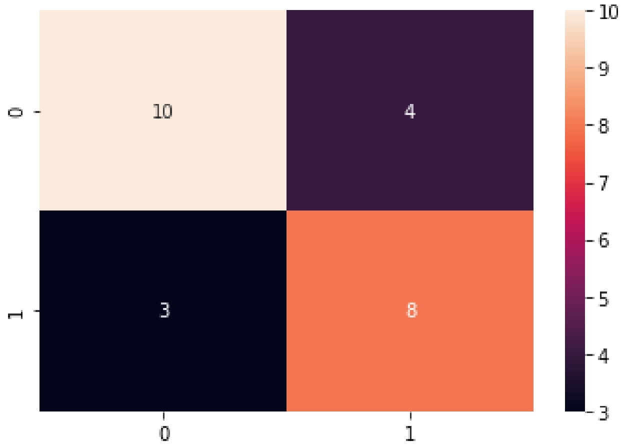
In [34]:

```
print(dt_cr)
print("\n\tDecision Tree")
sns.heatmap(dt_cm,annot=True)
plt.plot()
```

```
              precision    recall  f1-score   support

           0       0.77      0.71      0.74        14
           1       0.67      0.73      0.70        11

    accuracy                           0.72        25
   macro avg       0.72      0.72      0.72        25
weighted avg       0.72      0.72      0.72        25


 Decision Tree
```

Out[34]:

```
[]
```



Classification Report and Confusion Matrix for Random Forest

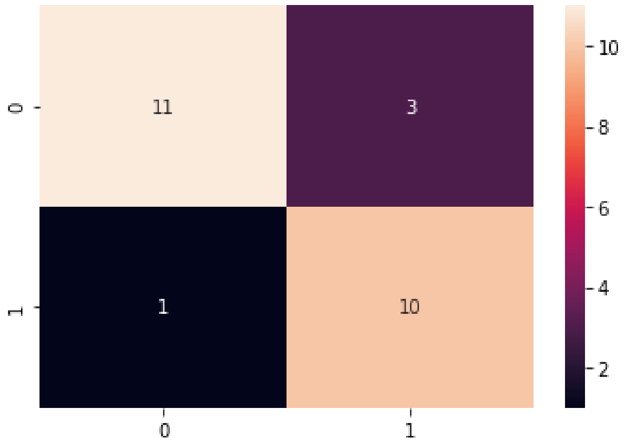
In [35]:

```
print(rf_cr)
print("\n\tRandom Forest")
sns.heatmap(rf_cm,annot=True)
plt.plot()
```

```
              precision    recall  f1-score   support

           0       0.92      0.79      0.85        14
           1       0.77      0.91      0.83        11

    accuracy                           0.84        25
   macro avg       0.84      0.85      0.84        25
weighted avg       0.85      0.84      0.84        25


 Random Forest
```

Out[35]:

```
[]
```



Testing with New Image

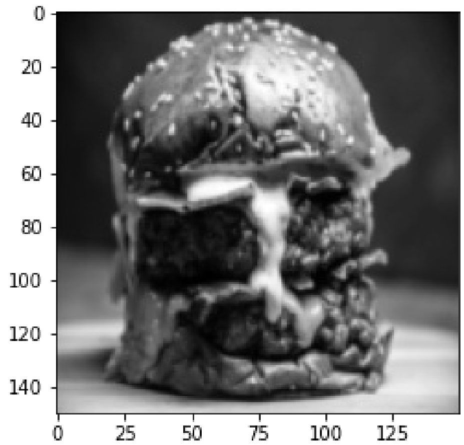
In [36]:

```
new_data=[]

url=input("image url: ")
img=skimage.io.imread(url,as_gray=True)
img=resize(img,(150,150))
new_data.append(np.ndarray.flatten(img))
print("\n\t[0] - Burger [1]- Pizza Slice\n")
print("Logistic Regression Model Output     : ",lgr_cv.predict([new_data[0]]))
print("Support Vector Machine Model Output : ",svm_cv.predict([new_data[0]]))
print("Decision Tree Model Output           : ",dtree.predict([new_data[0]]))
print("Random Forest Model Output           : ",rf.predict([new_data[0]]))
print("K Neighbors Model Output             : ",kn_model.predict([new_data[0]]))

plt.imshow(img,cmap="gray")
plt.plot()
```

```
image url: https://twisper.com/wp-content/uploads/2020/03/close-up-photo-of-burger-391590
6-scaled.jpg

 [0] - Burger [1]- Pizza Slice

Logistic Regression Model Output     :  [0]
Support Vector Machine Model Output :  [0]
Decision Tree Model Output           :  [0]
Random Forest Model Output           :  [0]
K Neighbors Model Output             :  [0]
```

Out[36]:

```
[]
```



In [37]:

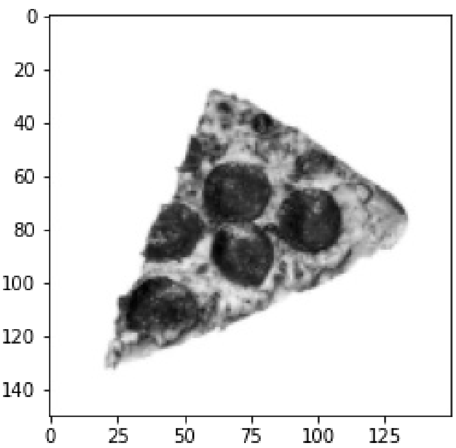
```
new_data=[]
```

```python
url=input("image url: ")
img=skimage.io.imread(url,as_gray=True)
img=resize(img,(150,150))
new_data.append(np.ndarray.flatten(img))
print("\n\t[0] - Burger [1]- Pizza Slice\n")
print("Logistic Regression Model Output     : ",lgr_cv.predict([new_data[0]]))
print("Support Vector Machine Model Output : ",svm_cv.predict([new_data[0]]))
print("Decision Tree Model Output           : ",dtree.predict([new_data[0]]))
print("Random Forest Model Output           : ",rf.predict([new_data[0]]))
print("K Neighbors Model Output             : ",kn_model.predict([new_data[0]]))

plt.imshow(img,cmap="gray")
plt.plot()
```

```
image url: https://media.istockphoto.com/photos/slice-of-fresh-italian-classic-original-p
epperoni-pizza-isolated-picture-id496546118?k=6&m=496546118&s=612x612&w=0&h=lQqZS7Wce2bVC
kgctEvBPBR5tlUxHGq2HjgpC7iOt3s=

 [0] - Burger [1]- Pizza Slice

Logistic Regression Model Output     :  [1]
Support Vector Machine Model Output :  [1]
Decision Tree Model Output           :  [1]
Random Forest Model Output           :  [1]
K Neighbors Model Output             :  [1]
```

Out[37]:

```
[]
```



In [ ]: