In [92]:
```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pylab as plt

from sklearn import model_selection
from sklearn.model_selection import train_test_split, KFold, cross_val_score, StratifiedKFold, learning_curv
from sklearn.metrics import confusion_matrix, make_scorer, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score
from sklearn.pipeline import Pipeline
from keras import models
model = models.Sequential()

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from keras.utils import to_categorical
from scikeras.wrappers import KerasClassifier, KerasRegressor
```

In [93]:
```
#loading images and their labels
X = np.load('X.npy') # images
Y = np.load('Y.npy') # labels for the images (0 = no IDC, 1 = IDC)
```

In [94]:
```
#making sure the data for X crossed over correctly
print(X[:3])
```

```
[[[226 164 206]
  [224 154 196]
  [225 175 211]
  ...
  [240 221 237]
  [232 184 214]
  [243 213 235]]

 [[217 142 188]
  [221 130 179]
  [224 150 196]
  ...
  [227 170 204]
  [229 180 215]
  [236 212 232]]

 [[237 178 212]
  [229 157 199]
  [218 125 175]
```

In [95]:
```
#making sure the data for Y crosssed over correctly
print(Y[:5])
```

```
[0 0 0 0 0]
```

In [96]:
```
#total number of images
print('Total number of images: {}'.format(len(X)))
```

```
Total number of images: 5547
```

In [97]:
```
#total number of negative IDC images
print('Number of negative IDC Images: {}'.format(np.sum(Y==0)))
```

```
Number of negative IDC Images: 2759
```

In [98]:
```
#total number of positive IDC images
print('Number of positive IDC Images: {}'.format(np.sum(Y==1)))
```

```
Number of positive IDC Images: 2788
```

In [99]:
```
#shape of the images
print('Image shape (Width, Height, Channels): {}'.format(X[0].shape))
```

```
Image shape (Width, Height, Channels): (50, 50, 3)
```

In [100]:
```
#train/test split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

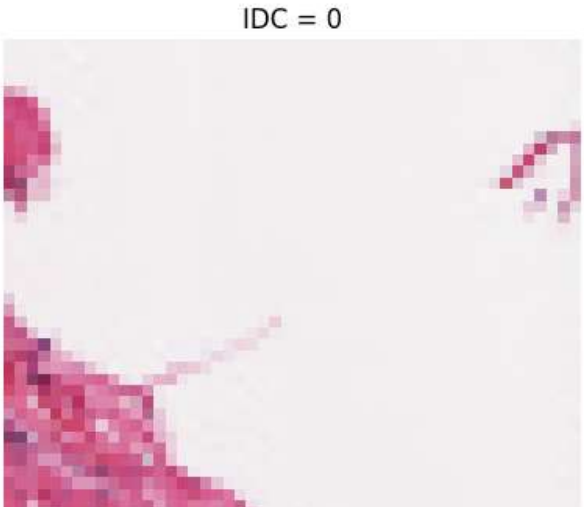
In [101]:
```
# Reduce Sample Size
x_train = x_train[0:30000]
y_train = y_train[0:30000]
x_test = x_test[0:30000]
y_test = y_test[0:30000]
```

In [102]:
```
# rescale pizel intensity
x_train = x_train / 256.0
x_test = x_test / 256.0
```

In [103]:
```
#verifying shape
print("Training Data Shape:", x_train.shape)
print("Testing Data Shape:", x_test.shape)
```

```
Training Data Shape: (4437, 50, 50, 3)
Testing Data Shape: (1110, 50, 50, 3)
```

In [104]:
```
#displaying the first five images in the training set along with the labels
for i in range(5):
    plt.imshow(x_train[i], cmap='gray'), plt.axis("off")
    plt.title('IDC = %d'%y_train[i])
    plt.show()
```



In [105]:
```
#printing the data to show that it's now 0-1
print(x_train[:3])
```

```
[[[[0.9453125  0.94140625 0.94921875]
   [0.953125   0.92578125 0.94140625]
   [0.953125   0.94140625 0.9375    ]
   ...
   [0.94921875 0.9375     0.9453125 ]
   [0.9453125  0.94921875 0.953125  ]
   [0.94921875 0.9296875  0.93359375]]

  [[0.95703125 0.92578125 0.93359375]
   [0.9375     0.94140625 0.93359375]
   [0.95703125 0.92578125 0.94140625]
   ...
   [0.94921875 0.9375     0.9453125 ]
   [0.9609375  0.93359375 0.9453125 ]
   [0.9453125  0.953125   0.94140625]]

  [[0.94921875 0.9375     0.9453125 ]
   [0.953125   0.9296875  0.95703125]
   [0.94921875 0.93359375 0.94140625]
```

In [106]:
```
# reshape data

x_train_r = x_train.reshape(x_train.shape[0], x_train.shape[1]*x_train.shape[2]*x_train.shape[3])
x_test_r = x_test.reshape(x_test.shape[0], x_test.shape[1]*x_test.shape[2]*x_test.shape[3])

print("x_train shape: ",x_train_r.shape)
print("x_test shape: ",x_test_r.shape)
```

```
x_train shape:  (4437, 7500)
x_test shape:  (1110, 7500)
```

In [107]:
```
def buildclassifier():
    classifier = Sequential() # initialize neural network
    classifier.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu', input_dim = x_train
    classifier.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
    classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier
```

In [108]:
```
classifier = KerasClassifier(build_fn = buildclassifier, epochs = 200)
accuracies = cross_val_score(estimator = classifier, X = x_train_r, y = y_train, cv = 6)
mean = accuracies.mean()
variance = accuracies.std()
```

```
116/116 [==============================] - 1s 6ms/step - loss: 0.5137 - accuracy: 0.7591
Epoch 15/200
116/116 [==============================] - 1s 6ms/step - loss: 0.5236 - accuracy: 0.7491
Epoch 16/200
116/116 [==============================] - 1s 7ms/step - loss: 0.5227 - accuracy: 0.7493
Epoch 17/200
116/116 [==============================] - 1s 6ms/step - loss: 0.5052 - accuracy: 0.7691
Epoch 18/200
116/116 [==============================] - 1s 6ms/step - loss: 0.5080 - accuracy: 0.7647
Epoch 19/200
116/116 [==============================] - 1s 7ms/step - loss: 0.5178 - accuracy: 0.7572
Epoch 20/200
116/116 [==============================] - 1s 7ms/step - loss: 0.5187 - accuracy: 0.7555
Epoch 21/200
116/116 [==============================] - 1s 6ms/step - loss: 0.5014 - accuracy: 0.7699
Epoch 22/200
116/116 [==============================] - 1s 6ms/step - loss: 0.5076 - accuracy: 0.7610
Epoch 23/200
116/116 [==============================] - 1s 6ms/step - loss: 0.5124 - accuracy: 0.7593
Epoch 24/200
```

In [109]:
```
print("Accuracy mean: "+ str(mean))
print("Accuracy variance: "+ str(variance))
```

```
Accuracy mean: 0.7189570152019407
Accuracy variance: 0.027899708347895104
```

In [110]:
```
#classifiying decision tree and fitting it
dtc = DecisionTreeClassifier()
dtc_y = dtc.fit(x_train_r,y_train)
```

In [111]:
```
dtc_pred = dtc_y.predict(x_test_r)
print('Accuracy Score : ' + str(accuracy_score(y_test,dtc_pred)))
print('Precision Score : ' + str(precision_score(y_test,dtc_pred)))
print('Recall Score : ' + str(recall_score(y_test,dtc_pred)))
print('F1 Score : ' + str(f1_score(y_test,dtc_pred)))
```

```
Accuracy Score : 0.6855855855855856
Precision Score : 0.7064220183486238
Recall Score : 0.6707317073170732
F1 Score : 0.6881143878462913
```

In [112]:
```
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,dtc_pred)))
```

```
Confusion Matrix :
[[376 160]
 [189 385]]
```

In [ ]:
```

```

In [113]:
```
#classifying random forest tree and fitting it
rfc= RandomForestClassifier(n_estimators = 100, random_state=42)
rfc_y = rfc.fit(x_train_r,y_train)
```

In [114]:
```
rfc_pred = rfc_y.predict(x_test_r)
print('Accuracy Score : ' + str(accuracy_score(y_test,rfc_pred)))
print('Precision Score : ' + str(precision_score(y_test,rfc_pred)))
print('Recall Score : ' + str(recall_score(y_test,rfc_pred)))
print('F1 Score : ' + str(f1_score(y_test,rfc_pred)))
```

```
Accuracy Score : 0.7747747747747747
Precision Score : 0.7852112676056338
Recall Score : 0.7770034843205574
F1 Score : 0.7810858143607706
```

In [115]:
```
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,rfc_pred)))
```

```
Confusion Matrix :
[[414 122]
 [128 446]]
```

In [ ]:
```

```

In [116]:
```
#SVC classiying and fitting
svc = SVC(random_state=42)
svc_y = svc.fit(x_train_r,y_train)
```

In [117]:
```
svc_pred = svc_y.predict(x_test_r)
print('Accuracy Score : ' + str(accuracy_score(y_test,svc_pred)))
print('Precision Score : ' + str(precision_score(y_test,svc_pred)))
print('Recall Score : ' + str(recall_score(y_test,svc_pred)))
print('F1 Score : ' + str(f1_score(y_test,svc_pred)))
```

```
Accuracy Score : 0.7864864864864864
Precision Score : 0.8025134649910234
Recall Score : 0.7787456445993032
F1 Score : 0.790450928381963
```

In [118]:
```
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,svc_pred)))
```

```
Confusion Matrix :
[[426 110]
 [127 447]]
```

In [ ]:
```

```

In [119]:
```
#logistic regression classifying and fittnig
lr = LogisticRegression()
lr_y = lr.fit(x_train_r,y_train)
```

```
C:\Users\brean\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

In [120]:
```
lr_pred = lr_y.predict(x_test_r)
print('Accuracy Score : ' + str(accuracy_score(y_test,lr_pred)))
print('Precision Score : ' + str(precision_score(y_test,lr_pred)))
print('Recall Score : ' + str(recall_score(y_test,lr_pred)))
print('F1 Score : ' + str(f1_score(y_test,lr_pred)))
```

```
Accuracy Score : 0.7018018018018019
Precision Score : 0.7237569060773481
Recall Score : 0.6846689895470384
F1 Score : 0.7036705461056401
```

In [121]:
```
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,lr_pred)))
```

```
Confusion Matrix :
[[386 150]
 [181 393]]
```

In [ ]:
```

```

In [122]:
```
#logistic regression classifying and fittnig
knn = KNeighborsClassifier()
knn_y = knn.fit(x_train_r,y_train)
```

In [123]:
```
#logisitic regression accuracy
kscore = knn.score(x_test_r,y_test)
print("KNeighbors accuracy", kscore)
```

```
KNeighbors accuracy 0.7117117117117117
```

In [124]:
```
knn_pred = knn_y.predict(x_test_r)
print('Accuracy Score : ' + str(accuracy_score(y_test,knn_pred)))
print('Precision Score : ' + str(precision_score(y_test,knn_pred)))
print('Recall Score : ' + str(recall_score(y_test,knn_pred)))
print('F1 Score : ' + str(f1_score(y_test,knn_pred)))
```

```
Accuracy Score : 0.7117117117117117
Precision Score : 0.7702127659574468
Recall Score : 0.6306620209059234
F1 Score : 0.6934865900383141
```

In [125]:
```
print('Confusion Matrix : \n' + str(confusion_matrix(y_test,lr_pred)))
```

```
Confusion Matrix :
[[386 150]
 [181 393]]
```

In [ ]:
```
```

In [126]:
```
results = []
results.append(mean)
results.append(dscore)
results.append(rscore)
results.append(sscore)
results.append(lscore)
results.append(kscore)
print(results)
```

```
[0.7189570152019407, 0.6891891891891891, 0.7738738738738739, 0.7801801801801802, 0.69009009009009, 0.711711
7117117117]
```

In [127]:
```
models = []
models.append(('classifier', KerasClassifier()))
models.append(('dtc', DecisionTreeClassifier()))
models.append(('rfc', RandomForestClassifier()))
models.append(('svc', SVC()))
models.append(('ls', LogisticRegression()))
models.append(('knn', KNeighborsClassifier()))
print(models)
```

```
[('classifier', KerasClassifier(
        model=None
        build_fn=None
        warm_start=False
        random_state=None
        optimizer=rmsprop
        loss=None
        metrics=None
        batch_size=None
        validation_batch_size=None
        verbose=1
        callbacks=None
        validation_split=0.0
        shuffle=True
        run_eagerly=False
        epochs=1
        class_weight=None
)), ('dtc', DecisionTreeClassifier()), ('rfc', RandomForestClassifier()), ('svc', SVC()), ('ls', LogisticRe
gression()), ('knn', KNeighborsClassifier())]
```

In [ ]:
```
```

In [ ]:
```
```

```
The below code is something I was working on; the gridsearch. But, couldn't get it to work for this
project.
```

```
minmax = MinMaxScaler()
```

```
dtc = DecisionTreeClassifier()
rfc= RandomForestClassifier(n_estimators = 100, random_state=42)
svc = SVC(random_state=42)
lr = LogisticRegression()
knn = KNeighborsClassifier()
dtr = DecisionTreeRegressor()
```

```
```

```
pipe = Pipeline(steps = [('scaler', minmax), ('classifier', dtr)])
```

```
pipe.fit(x_train_r, y_train)
```

```
param_grid = [{'classifier__max_depth': [2,6,8,10],
               'classifier__min_samples_split': [2,5,10,15]},
               {'classifier':[dtc],
                'classifier__max_depth': [2,6,8,10],
               'classifier__min_samples_split': [2,5,10,15],
               'classifier__max_leaf_nodes': [None,10,20,50,100]},
               {'classifier':[rfc],
               'classifier__max_depth': [2,6,8,10],
               'classifier__min_samples_split': [2,5,10,15],
               'classifier__max_features': [2,3,4,5,6]},
               {'classifier':[svc],
               'classifier__max_depth': [2,6,8,10],
               'classifier__min_samples_split': [2,5,10,15],
               'classifier__max_features': [2,3,4,5,6]},
                {'classifier':[lr],
               'classifier__max_depth': [2,6,8,10],
               'classifier__min_samples_split': [2,5,10,15],
               'classifier__max_features': [2,3,4,5,6]},
                {'classifier':[knn],
               'classifier__max_depth': [2,6,8,10],
               'classifier__min_samples_split': [2,5,10,15],
               'classifier__max_features': [2,3,4,5,6]},
               {'classifier':[dtr],
               'classifier__max_depth': [2,6,8,10],
               'classifier__min_samples_split': [2,5,10,15],
               'classifier__max_features': [2,3,4,5,6]},
              ]
```

```
grid_search = GridSearchCV(pipe, param_grid, cv = 5, verbose = 0)
```

```
best_model = grid_search.fit(x_train_r, y_train)
```