In [1]:
```
import numpy as np
import pandas as pd
import os
import cv2 as cv
import matplotlib.pyplot as plt
import seaborn as sns
```

# Load and Check Data

In [2]:
```
all_0 = "E:\\lukemia\\C-NMC_Leukemia\\training_data\\fold_0\\all"
all_1 = "E:\\lukemia\\C-NMC_Leukemia\\training_data\\fold_1\\all"
all_2 = "E:\\lukemia\\C-NMC_Leukemia\\training_data\\fold_2\\all"

hem_0 = "E:\\lukemia\\C-NMC_Leukemia\\training_data\\fold_0\\hem"
hem_1 = "E:\\lukemia\\C-NMC_Leukemia\\training_data\\fold_1\\hem"
hem_2 = "E:\\lukemia\\C-NMC_Leukemia\\training_data\\fold_2\\hem"
```

In [3]:
```
def get_path_image(folder):
    image_paths = []
    image_fnames = os.listdir(folder)
    for img_id in range(len(image_fnames)):
        img = os.path.join(folder,image_fnames[img_id])
        image_paths.append(img)

    return image_paths
```

In [4]:
```
img_data = []

for i in [all_0,all_1,all_2,hem_0,hem_1,hem_2]:
    paths = get_path_image(i)
    img_data.extend(paths)
print(len(img_data))
```

```
10661
```

In [5]:
```
data = {"img_data":img_data,
        "labels":[np.nan for x in range(len(img_data))]}

data = pd.DataFrame(data)
```

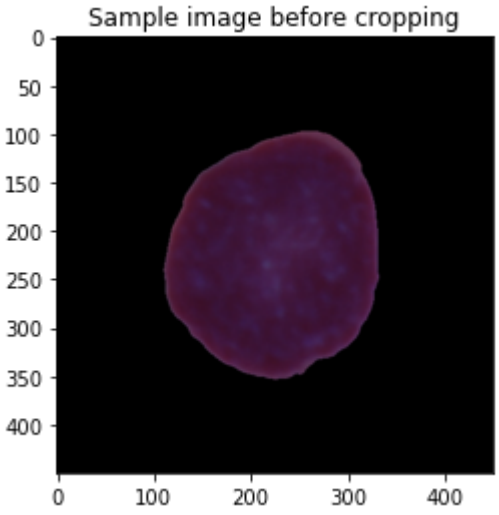
In [6]:
```
data["labels"][0:7272] = 1
data["labels"][7272:10661] = 0
```

```
<ipython-input-6-1c691c1b7beb>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data["labels"][0:7272] = 1 # ALL
<ipython-input-6-1c691c1b7beb>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data["labels"][7272:10661] = 0 # HEM
```

In [7]:
```
data["labels"] = data["labels"].astype("int64")
```

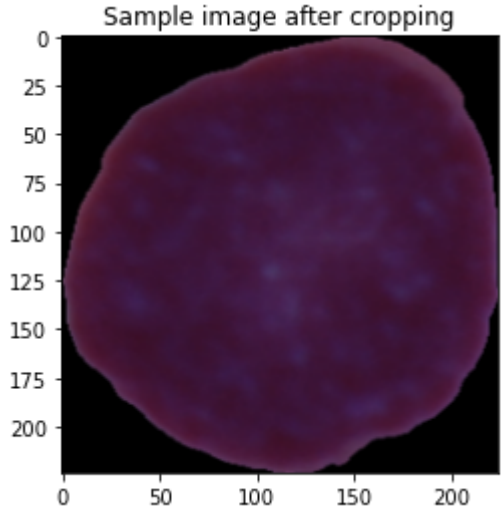
# Crop Black Edges In Image

In [8]:
```
image = cv.imread(data["img_data"][1000])
plt.imshow(image)
plt.title("Sample image before cropping")
plt.show()
```



In [9]:
```
img_list = []
for i in range(len(img_data)):
    image = cv.imread(data["img_data"][i])
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    thresh = cv.threshold(gray, 0, 255, cv.THRESH_BINARY_INV + cv.THRESH_OT

    result = cv.bitwise_and(image, image, mask=thresh)
    result[thresh==0] = [255,255,255]
    (x, y, z_) = np.where(result > 0)
    mnx = (np.min(x))
    mxx = (np.max(x))
    mny = (np.min(y))
    mxy = (np.max(y))
    crop_img = image[mnx:mxx,mny:mxy,:]
    crop_img_r = cv.resize(crop_img, (224,224))
    img_list.append(crop_img_r)
```

In [10]:
```
plt.imshow(img_list[1000])
plt.title("Sample image after cropping")
plt.show()
```




# Feature Extraction with VGG19, ResNet50 or ResNet101

In [11]:
```
from tensorflow.keras.applications import ResNet50, ResNet101
from keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import preprocess_input
```

In [12]:
```
def feature_extract(model):
    if model == "VGG19": model = VGG19(weights='imagenet',include_top=False
    elif model == "ResNet50": model = ResNet50(weights='imagenet',include_t
    elif model == "ResNet101": model = ResNet101(weights='imagenet',include_
    return model
```

In [13]:
```
model = feature_extract("ResNet50") # or "VGG19", "ResNet101"
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5)
94765736/94765736 [==============================] - 8s 0us/step
```

In [14]:
```
features_list = []
for i in range(len(img_list)):

    image = img_list[i].reshape(-1, 224, 224, 3)
    image = preprocess_input(image)
    features = model.predict(image).reshape(2048,)

    features_list.append(features)
```

```
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 1s 793ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 859ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 934ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 837ms/step
1/1 [==============================] - 1s 875ms/step
1/1 [==============================] - 1s 964ms/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 1s 901ms/step
1/1 [==============================] - 1s 1s/step
```

In [15]:
```
features_df = pd.DataFrame(features_list)
```

In [16]:
```
features_df["labels"] = data["labels"]
```

In [17]:
```
x = features_df.drop(['labels'], axis = 1)
y = features_df.loc[:,"labels"].values
```

In [18]:
```
x
```

Out[18]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 5.253779 | 0.042580 | 0.051578 | 0.008029 | 0.057129 | 0.004375 | 0.227440 | 0.003590 | 0.005 |
| **1** | 8.079419 | 0.011574 | 0.053684 | 0.186904 | 0.000000 | 0.000000 | 0.042724 | 0.000000 | 0.574 |
| **2** | 6.266928 | 0.219212 | 0.018836 | 0.008475 | 0.000000 | 0.000000 | 0.224284 | 0.000000 | 0.000 |
| **3** | 6.486920 | 0.000000 | 0.020409 | 0.164188 | 0.000000 | 0.039742 | 0.098523 | 0.000000 | 0.136 |
| **4** | 5.148220 | 0.060940 | 0.000000 | 0.106592 | 0.000000 | 0.036498 | 0.074916 | 0.095883 | 0.034 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **10656** | 5.135458 | 0.023519 | 0.000000 | 0.013403 | 0.095962 | 0.044262 | 0.067926 | 0.006757 | 0.088 |
| **10657** | 5.458831 | 0.028126 | 0.000000 | 0.173168 | 0.000000 | 0.006622 | 0.014662 | 0.024785 | 0.046 |
| **10658** | 5.933911 | 0.000000 | 0.000000 | 0.069616 | 0.000000 | 0.110708 | 0.000000 | 0.000000 | 0.008 |
| **10659** | 5.647943 | 0.059169 | 0.000000 | 0.116159 | 0.007541 | 0.000000 | 0.112060 | 0.000000 | 0.081 |
| **10660** | 2.950781 | 0.067704 | 0.029904 | 0.096164 | 0.003031 | 0.000000 | 0.091273 | 0.000000 | 0.000 |

10661 rows × 2048 columns

In [19]:
```
print(f"Number of features before feature selection: {x.shape[1]}")
```

```
Number of features before feature selection: 2048
```

In [20]:
```
y
```

Out[20]:
```
array([1, 1, 1, ..., 0, 0, 0], dtype=int64)
```

# Data Scaling

In [21]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(x)
x_ = scaler.transform(x)
```

In [22]:
```
x_ = pd.DataFrame(x_)
```

# Feature Selection Methods

## ANOVA

In [23]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

def anova_fs():

    selector = SelectKBest(f_classif, k=500) # k is number of features
    selector.fit(x_, y)

    cols = selector.get_support(indices=True)
    anova_x = x_[cols]
    return anova_x
```

## Recursive Feature Elimination (RFE)

In [24]:
```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier

def RFE_fs():
    rfe_selector = RFE(estimator=RandomForestClassifier())
    rfe_selector.fit(x_, y)

    rfe_support = rfe_selector.get_support()
    rfe_feature = x_.loc[:,rfe_support].columns.tolist()

    rfe_x = x_[rfe_feature]
    return rfe_x
```

## Random Forest

In [25]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

def rf_fs():
    embeded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimato
    embeded_rf_selector.fit(x, y)

    embeded_rf_support = embeded_rf_selector.get_support()
    embeded_rf_feature = x.loc[:,embeded_rf_support].columns.tolist()

    rf_x = x[embeded_rf_feature]
    return rf_x
```

In [26]:
```
fs_x = rf_fs() # feature selection methods "anova_fs", "RFE_fs"
```

In [27]:
```
print(f"Number of features after feature selection: {fs_x.shape[1]}")
```

```
Number of features after feature selection: 624
```

# Train Test Split

In [28]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(fs_x, y, test_size = 0.
```

# Classification with ML Algorithms

In [29]:
```
from sklearn.model_selection import cross_val_score,cross_val_predict
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score,precision_score,recall_score,accuracy_
from sklearn.model_selection import GridSearchCV
```

## kNN

In [30]:
```
neig = np.arange(1, 25)
train_accuracy = []
test_accuracy = []

for i, k in enumerate(neig):

    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    prediction_ = knn.predict(x_test)
    train_accuracy.append(knn.score(x_train, y_train))
    test_accuracy.append(knn.score(x_test, y_test))

print("Best accuracy is {} with K = {}".format(np.max(test_accuracy),1+test_
```

```
Best accuracy is 0.8312236286919831 with K = 13
```

In [31]:
```
knn = KNeighborsClassifier(n_neighbors=17)
knn.fit(x_train,y_train)
predicted = knn.predict(x_test)
score = knn.score(x_test, y_test)
knn_score_ = np.mean(score)

print('Accuracy : %.3f' % (knn_score_))
```

```
Accuracy : 0.827
```

In [32]:
```
p=precision_score(y_test, predicted)
print('Precision : %.3f' % (p))

r=recall_score(y_test, predicted)
print('Recall : %.3f' % (r))

f1=f1_score(y_test, predicted)
print('F1-score: %.3f' % (f1))

f1_w=f1_score(y_test, predicted, average='weighted')
print('Weighted f1-score: %.3f' % (f1_w))
```

```
Precision : 0.829
Recall : 0.942
F1-score: 0.882
Weighted f1-score: 0.818
```

In [33]:
```
cf_matrix = confusion_matrix(y_test, predicted)
sns.heatmap(cf_matrix, cmap="PuBu", annot=True, fmt='.0f')
plt.show()
```

# SVM

In [34]:
```
param_grid_svm = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf', 'poly']}

SVM_grid = GridSearchCV(svm.SVC(), param_grid_svm, cv=5)
SVM_grid.fit(x_train, y_train)
```

Out[34]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf', 'poly']})
```

In [35]:
```
print(SVM_grid.best_params_)

print(SVM_grid.best_estimator_)
```

```
{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
SVC(C=10, gamma=0.01)
```

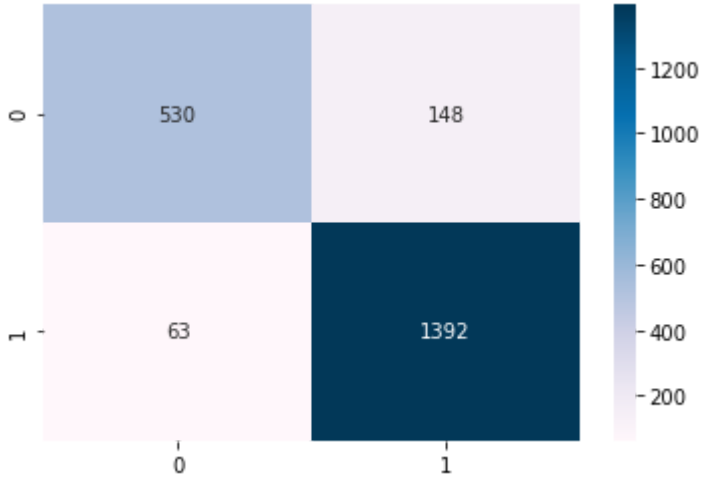
In [36]:
```
svm_clf = svm.SVC(C=100, gamma=0.01, kernel='rbf')
svm_clf.fit(x_train,y_train)
predicted = svm_clf.predict(x_test)
score = svm_clf.score(x_test, y_test)
svm_score_ = np.mean(score)

print('Accuracy : %.3f' % (svm_score_))
```

```
Accuracy : 0.901
```

In [37]:
```
p=precision_score(y_test, predicted)
print('precision : %.3f' % (p))

r=recall_score(y_test, predicted)
print('recall : %.3f' % (r))

f1=f1_score(y_test, predicted)
print('f1-score: %.3f' % (f1))

f1_w=f1_score(y_test, predicted, average='weighted')
print('weighted f1-score: %.3f' % (f1_w))
```

```
precision : 0.904
recall : 0.957
f1-score: 0.930
weighted f1-score: 0.899
```

In [38]:
```
cf_matrix = confusion_matrix(y_test, predicted)
sns.heatmap(cf_matrix, cmap="PuBu", annot=True, fmt='.0f')
plt.show()
```




## Random Forest

In [39]:
```
param_grid_rf = {
    'n_estimators': [200, 500],
    'max_depth' : [4,5,6,7,8]}

RF_grid = GridSearchCV(estimator=RandomForestClassifier(), param_grid=param_
RF_grid.fit(x_train, y_train)
```

Out[39]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [4, 5, 6, 7, 8],
                         'n_estimators': [200, 500]})
```

In [40]:
```
print(RF_grid.best_params_)
```

```
{'max_depth': 8, 'n_estimators': 500}
```

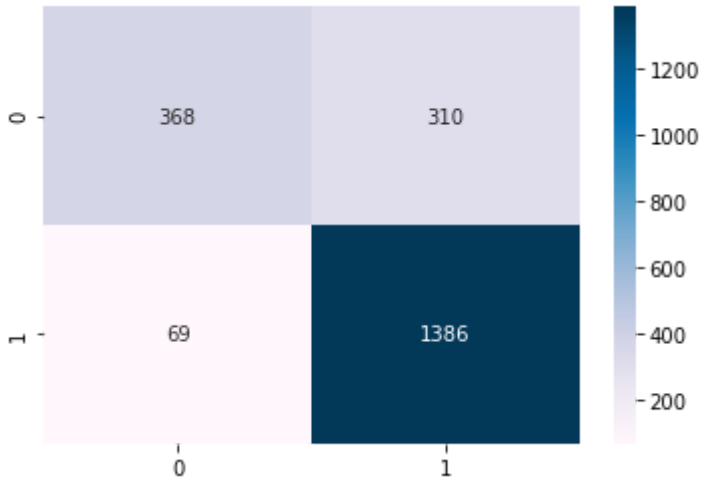
In [41]:
```
r_forest = RandomForestClassifier(500,max_depth=8, random_state=5)
r_forest.fit(x_train,y_train)
predicted = r_forest.predict(x_test)
score = r_forest.score(x_test, y_test)
rf_score_ = np.mean(score)

print('Accuracy : %.3f' % (rf_score_))
```

```
Accuracy : 0.822
```

In [42]:
```
p=precision_score(y_test, predicted)
print('precision : %.3f' % (p))

r=recall_score(y_test, predicted)
print('recall : %.3f' % (r))

f1=f1_score(y_test, predicted)
print('f1-score: %.3f' % (f1))

f1_w=f1_score(y_test, predicted, average='weighted')
print('weighted f1-score: %.3f' % (f1_w))
```

```
precision : 0.817
recall : 0.953
f1-score: 0.880
weighted f1-score: 0.810
```

In [43]:
```
cf_matrix = confusion_matrix(y_test, predicted)
sns.heatmap(cf_matrix, cmap="PuBu", annot=True, fmt='.0f')
plt.show()
```




## Naive Bayes

In [44]:
```
nb_model = GaussianNB()
nb_model.fit(x_train,y_train)
predicted = nb_model.predict(x_test)
score = nb_model.score(x_test, y_test)
nb_score_ = np.mean(score)

print('Accuracy : %.3f' % (nb_score_))
```

```
Accuracy : 0.762
```

In [45]:
```
p=precision_score(y_test, predicted)
print('precision : %.3f' % (p))

r=recall_score(y_test, predicted)
print('recall : %.3f' % (r))

f1=f1_score(y_test, predicted)
print('f1-score: %.3f' % (f1))

f1_w=f1_score(y_test, predicted, average='weighted')
print('weighted f1-score: %.3f' % (f1_w))
```

```
precision : 0.846
recall : 0.796
f1-score: 0.820
weighted f1-score: 0.765
```

In [46]:
```
cf_matrix = confusion_matrix(y_test, predicted)
sns.heatmap(cf_matrix, cmap="PuBu", annot=True, fmt='.0f')
plt.show()
```