Cataract Detection using KNN

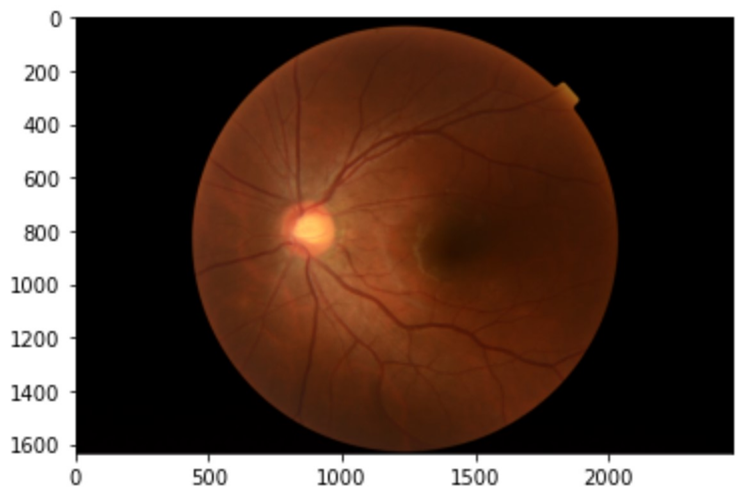
In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import os
```

In [2]:
```
DATADIR = 'Dataset'
# CATEGORIES = ['1_normal','2_cataract','2_glaucoma','3_retina_disease']
CATEGORIES = ['1_normal','2_cataract']
```

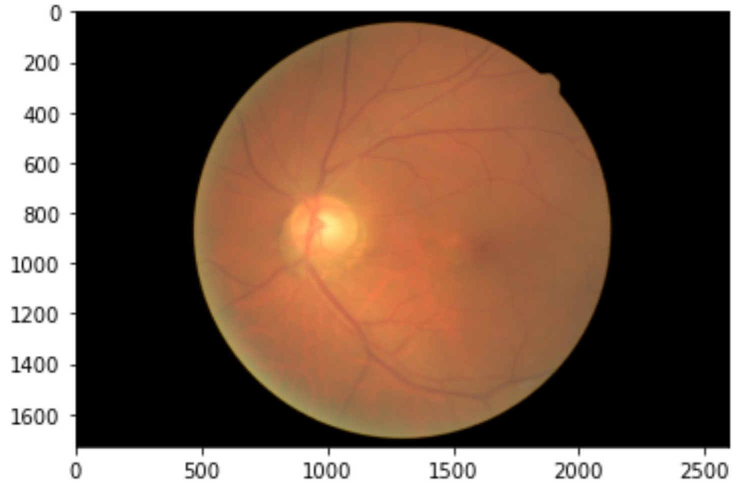
Displaying All Categories of Images

In [3]:
```
image = []
i = 0;
for category in CATEGORIES:
  path=os.path.join(DATADIR, category)
  print(path)
  for img in os.listdir(path):
    img_array=cv2.imread(os.path.join(path,img))
    img_array=cv2.cvtColor(img_array,cv2.COLOR_BGR2RGB)
    print(img_array.shape)
    image.append(img_array)
    plt.imshow(image[i])
    i += 1
    plt.show()
    break
```

```
Dataset\1_normal
(1632, 2464, 3)
```




```
Dataset\2_cataract
(1728, 2592, 3)
```

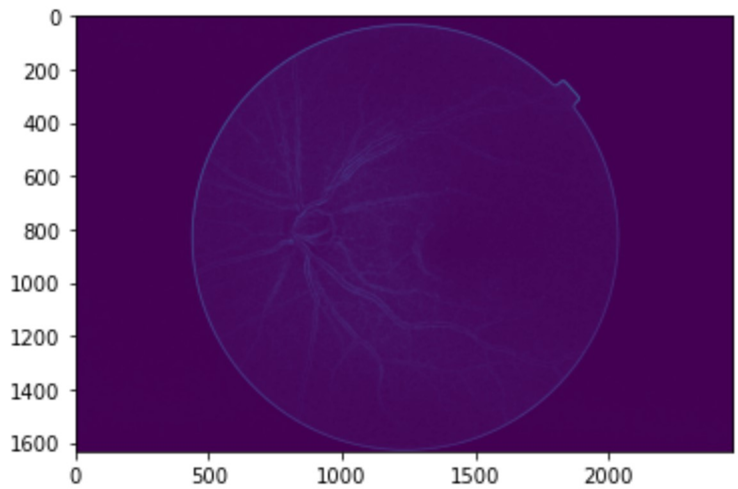


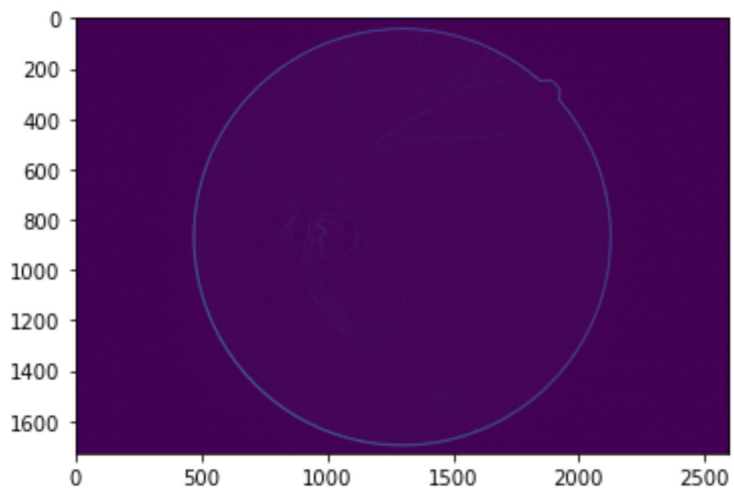
Creating Histogram for Images

In [4]:
```
def bulid_histogram(image) :
  image_bg = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  column,row = image_bg.shape
  # Calculate the histogram of the image
  hist, bins = np.histogram(image_bg.flatten(), 256, [0, 256])
  # Calculate the cumulative sum of the histogram
  cdf = hist.cumsum()
  cdf_normalized = cdf * hist.max() / cdf.max()
  # Apply histogram equalization to the image
  img_equalized = np.interp(image_bg.flatten(), bins[:-1], cdf_normalized).reshape(
  # Save the output image
  # cv2.imwrite('output_image.jpg', img_equalized)
  image_eq = img_equalized.copy()
  return image_eq
```

Applying Sobel Operator

In [5]:
```
def sobelOperator(image) :
  # Load the image
  img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  # Apply the Sobel operator in the x-direction
  sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
  # Apply the Sobel operator in the y-direction
  sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
  # Calculate the magnitude of the gradient
  mag = np.sqrt(sobelx**2 + sobely**2)
  # Normalize the magnitude image to range 0-255
  mag = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)
  return mag
for i in range(0,len(image)) :
  im = sobelOperator(image[i])
  plt.imshow(im)
  plt.show()
```





Preprocessing

In [6]:
```
IMG_SIZE = 400
training_data=[]
def create_training_data():
  for category in CATEGORIES:
    path=os.path.join(DATADIR, category)
    class_num=CATEGORIES.index(category)
    for img in os.listdir(path):
      try:
        img_array=cv2.imread(os.path.join(path,img))
        img_array = sobelOperator(img_array)
        new_array=cv2.resize(img_array,(IMG_SIZE,IMG_SIZE))
        training_data.append([new_array,class_num])
      except Exception as e:
        pass
create_training_data()
```

In [7]:
```
print(len(training_data))
```

400

In [8]:
```
lenofimage = len(training_data)
```

In [9]:
```
X=[]
y=[]
for categories, label in training_data:
  X.append(categories)
  y.append(label)
X= np.array(X).reshape(lenofimage,-1)
```

In [10]:
```
X.shape
X
```

Out[10]:
```
array([[2, 0, 0, ..., 0, 0, 1],
       [0, 2, 3, ..., 0, 1, 0],
       [2, 0, 0, ..., 0, 1, 1],
       ...,
       [1, 0, 2, ..., 2, 2, 1],
       [0, 1, 1, ..., 0, 1, 0],
       [1, 1, 2, ..., 1, 1, 2]], dtype=uint8)
```

In [11]:
```
X = X/255
X
```

Out[11]:
```
array([[0.00784314, 0.        , 0.        , ..., 0.        , 0.        ,
        0.00392157],
       [0.        , 0.00784314, 0.01176471, ..., 0.        , 0.00392157,
        0.        ],
       [0.00784314, 0.        , 0.        , ..., 0.        , 0.00392157,
        0.00392157],
       ...,
       [0.00392157, 0.        , 0.00784314, ..., 0.00784314, 0.00784314,
        0.00392157],
       [0.        , 0.00392157, 0.00392157, ..., 0.        , 0.00392157,
        0.        ],
       [0.00392157, 0.00392157, 0.00784314, ..., 0.00392157, 0.00392157,
        0.00784314]])
```

In [12]:
```
X[1]
```

Out[12]:
```
array([0.        , 0.00784314, 0.01176471, ..., 0.        , 0.00392157,
       0.        ])
```

In [13]:
```
y=np.array(y)
```

In [14]:
```
y.shape
```

Out[14]:
```
(400,)
```

Creating Model Using KNN

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier
# model = KNeighborsClassifier(n_neighbors=10,weights='distance',metric='minkowski'
model = KNeighborsClassifier(n_neighbors=6)
model.fit(X_train, y_train)
```

Out[16]:
```
KNeighborsClassifier(n_neighbors=6)
```

In [17]:
```
y2 = model.predict(X_test)
```

In [24]:
```
from sklearn.metrics import accuracy_score,classification_report
print("Accuracy on unknown data is",accuracy_score(y_test,y2))
```

```
Accuracy on unknown data is 0.75
```

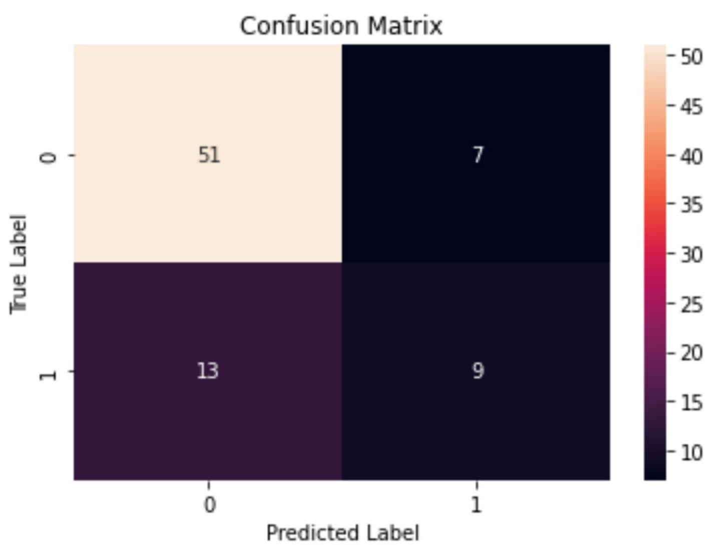
In [25]:
```
print("Accuracy on unknown data is",classification_report(y_test,y2))
```

```
Accuracy on unknown data is               precision    recall  f1-score   support

           0       0.80      0.88      0.84        58
           1       0.56      0.41      0.47        22

    accuracy                           0.75        80
   macro avg       0.68      0.64      0.65        80
weighted avg       0.73      0.75      0.74        80
```

In [19]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(y_test,y2)
sns.heatmap(cm, annot=True, fmt="d")
plt.title("Confusion Matrix")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.show()
```



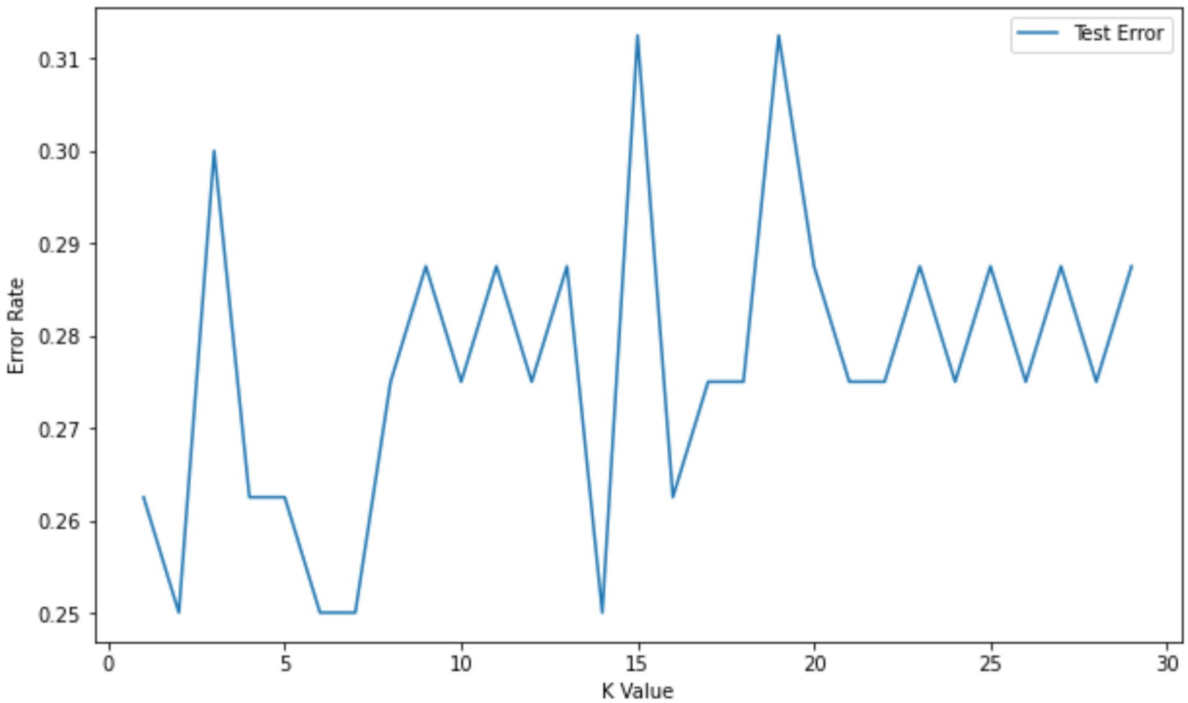
Elbow Method for Choosing Reasonable K values

In [20]:
```
test_error_rates = []
```

In [21]:
```
for k in range(1,30):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train,y_train)
    y_pred_test = knn_model.predict(X_test)
    test_error = 1 - accuracy_score(y_test,y_pred_test)
    test_error_rates.append(test_error)
```

In [29]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,30),test_error_rates,label='Test Error')
plt.legend()
plt.ylabel('Error Rate')
plt.xlabel('K Value')
```

Out[29]:
```
Text(0.5, 0, 'K Value')
```



In [ ]: