Blindness Detection

Import Libraries

In [2]:
```
import os, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import skimage.io
from skimage.transform import resize
from imgaug import augmenters as iaa
from tqdm import tqdm
import PIL
from PIL import Image, ImageOps
import cv2
from sklearn.utils import class_weight, shuffle
from keras.losses import binary_crossentropy
from keras.applications.resnet50 import preprocess_input
import keras.backend as K
import tensorflow as tf
from sklearn.metrics import f1_score, fbeta_score
from keras.utils import Sequence
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split

WORKERS = 2
CHANNEL = 3

import warnings
warnings.filterwarnings("ignore")
IMG_SIZE = 512
NUM_CLASSES = 5
SEED = 77
TRAIN_NUM = 1000 # use 1000 when you just want to explore new idea, use -1 for full train
```

```
Using TensorFlow backend.
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:516: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)
type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:517: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)
type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:518: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)
type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)
type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)
type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/opt/conda/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)
type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:545: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/opt/conda/lib/python3.6/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

Read data

In [3]:
```
df_train = pd.read_csv('../input/aptos2019-blindness-detection/train.csv')
df_test = pd.read_csv('../input/aptos2019-blindness-detection/test.csv')
```

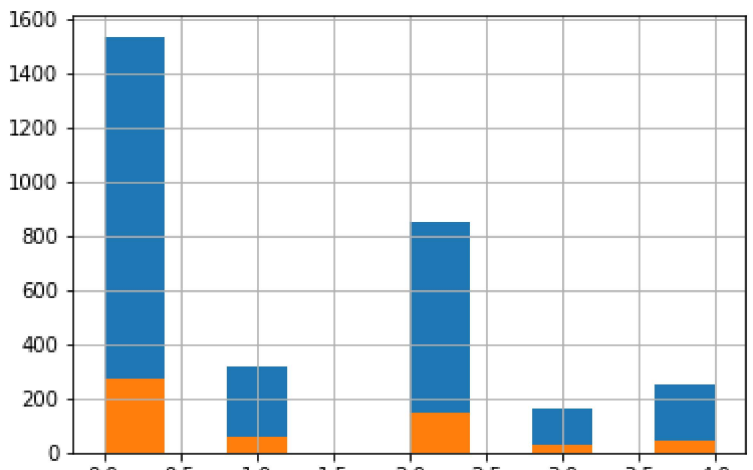
```
x = df_train['id_code']
y = df_train['diagnosis']

x, y = shuffle(x, y, random_state=SEED)
```

Checks the distribution of classes in both the training and validation sets using histograms.

The stratify=y parameter ensures that the class distribution of the original dataset (y) is maintained in both the training and validation sets. This is important in classification problems to avoid biased model training or evaluation due to imbalanced class distributions.

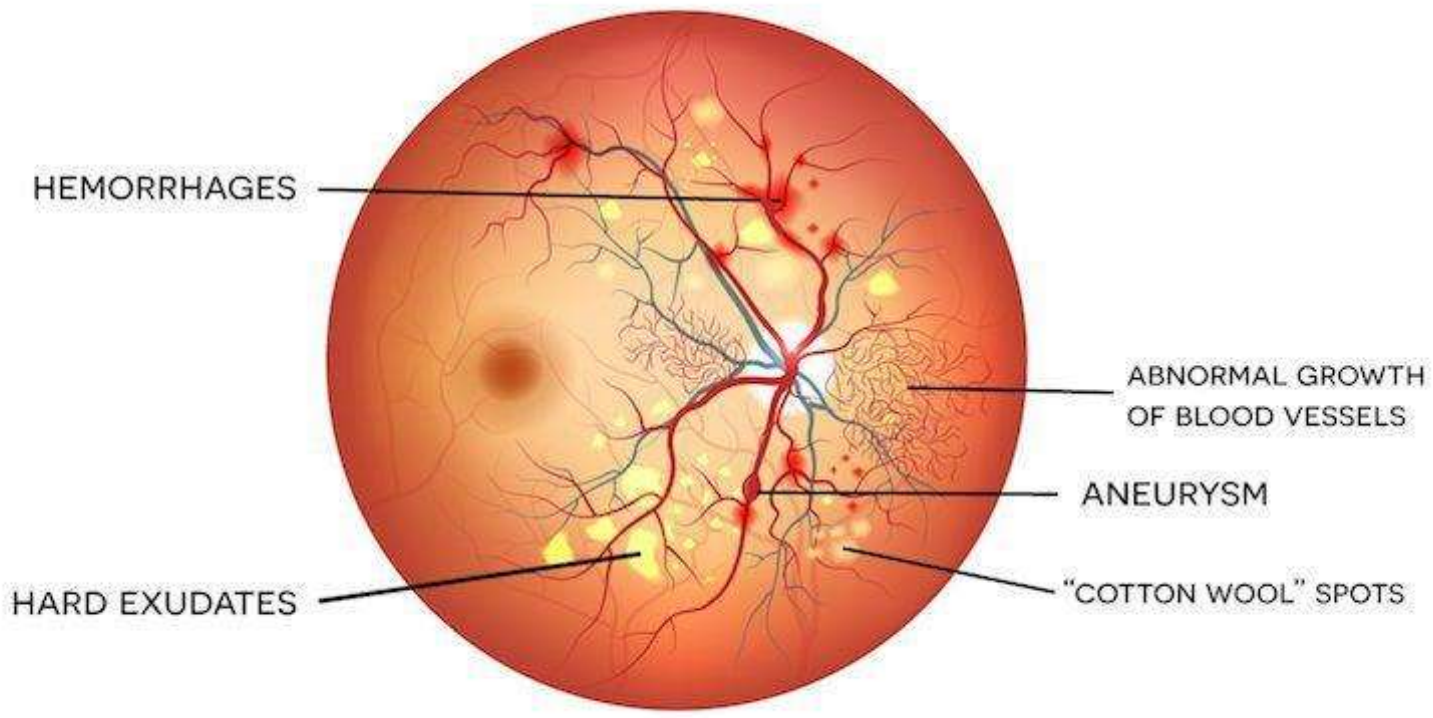
In [4]:
```
train_x, valid_x, train_y, valid_y = train_test_split(x, y, test_size=0.15,
                                                      stratify=y, random_state=SEED)
print(train_x.shape, train_y.shape, valid_x.shape, valid_y.shape)
train_y.hist()
valid_y.hist()
```

Out[4]:
```
(3112,) (3112,) (550,) (550,)
<matplotlib.axes._subplots.AxesSubplot at 0x7e5b8ffad470>
```



Explain Diabetic Retinopathy

How do we know that a patient have diabetic retinopahy? There are at least 5 things to spot on

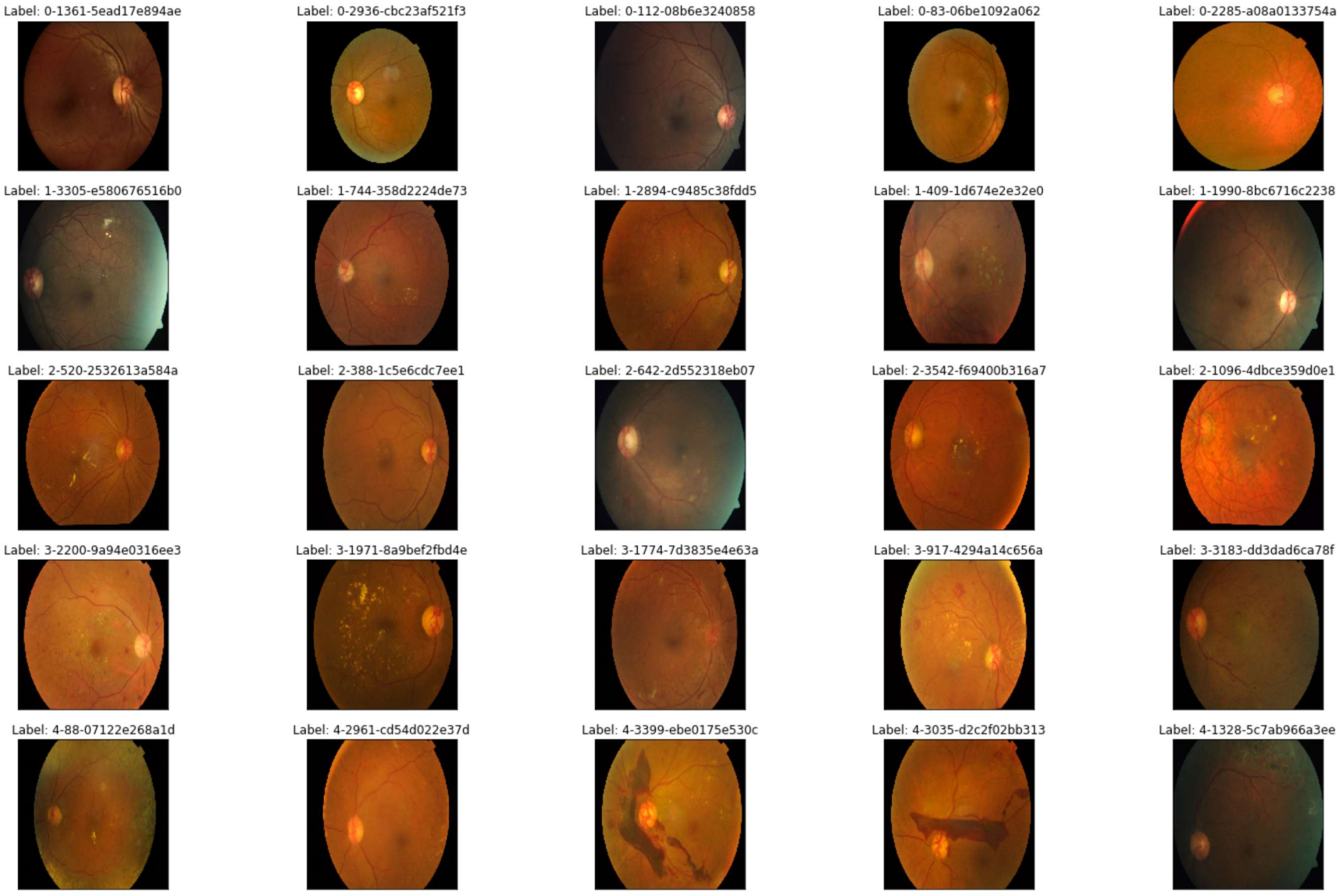


Original Inputs

First, let's take a glance at the original inputs. Each row represents a different severity level. Two issues make it difficult to discern the severity accurately. Firstly, some images appear very dark (pic(0,2) and pic(4,4)), while at times, varying color illuminations can be confusing (pic(3,3)). Secondly, certain pictures contain uninformative dark areas (pic(0,1), pic(0,3)). This becomes particularly crucial when reducing the picture size, as informative areas may become too small. Hence, it seems intuitive to crop out these uninformative areas in the latter case.

In [5]:
```
%%time
fig = plt.figure(figsize=(25, 16))
# display 10 images from each class
for class_id in sorted(train_y.unique()):
    for i, (idx, row) in enumerate(df_train.loc[df_train['diagnosis'] == class_id].sample(5, random_state=SEED).iterrow
        ax = fig.add_subplot(5, 5, class_id * 5 + i + 1, xticks=[], yticks=[])
        path=f"../input/aptos2019-blindness-detection/train_images/{row['id_code']}.png"
        image = cv2.imread(path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))

        plt.imshow(image)
        ax.set_title('Label: %d-%d-%s' % (class_id, idx, row['id_code']) )
```

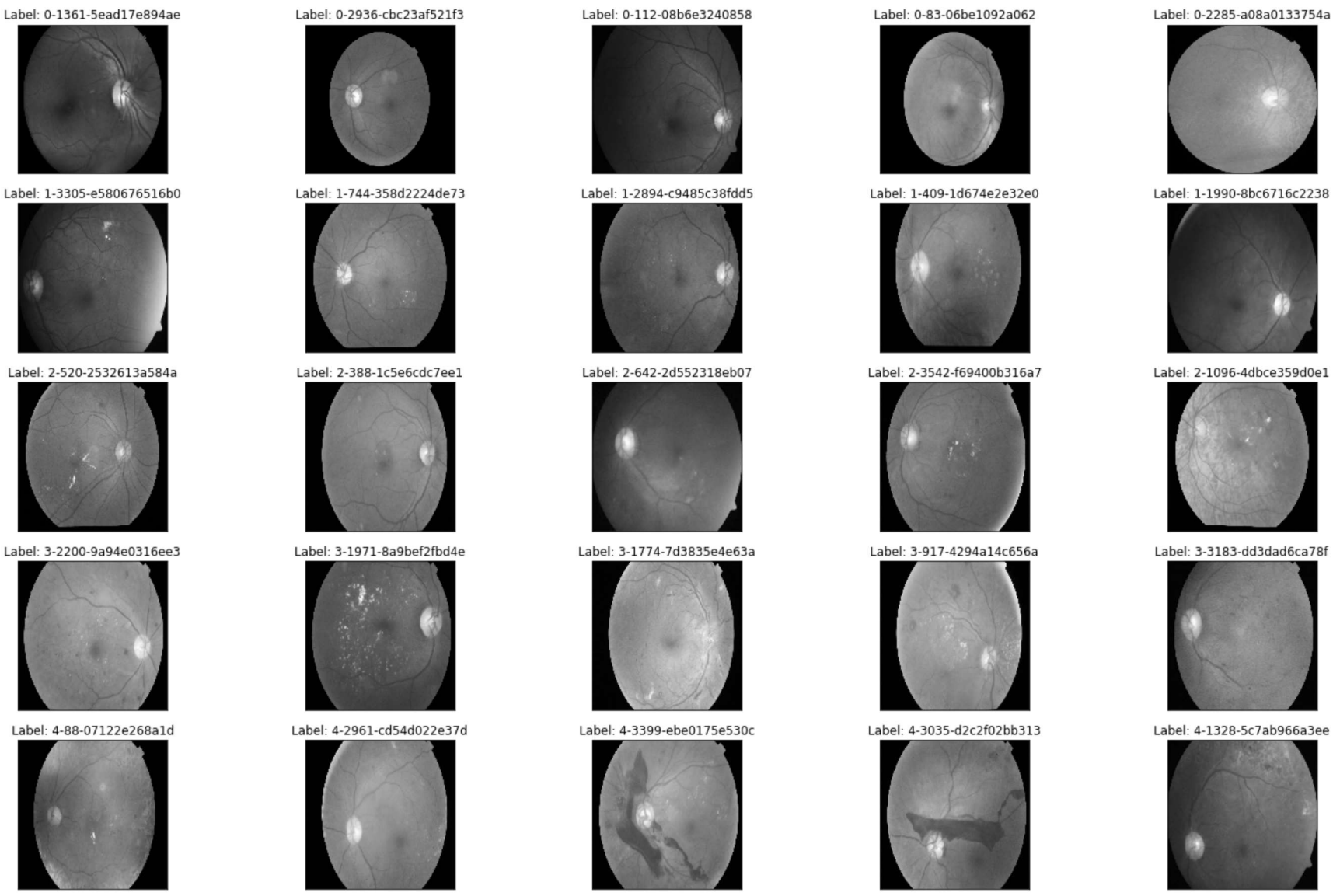
```
CPU times: user 3.06 s, sys: 259 ms, total: 3.31 s
Wall time: 4.4 s
```



We can try gray scale and feel understand better for some pictures, as color distraction is gone. For example, we can see more blood clearer in the upper part of pic(4,4), which has severity of level 4.

In [6]:
```
%%time
fig = plt.figure(figsize=(25, 16))
for class_id in sorted(train_y.unique()):
    for i, (idx, row) in enumerate(df_train.loc[df_train['diagnosis'] == class_id].sample(5, random_state=SEED).iterrow
        ax = fig.add_subplot(5, 5, class_id * 5 + i + 1, xticks=[], yticks=[])
        path=f"../input/aptos2019-blindness-detection/train_images/{row['id_code']}.png"
        image = cv2.imread(path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
#         image=cv2.addWeighted ( image, 0 , cv2.GaussianBlur( image , (0 ,0 ) , 10) ,-4 ,128)
        image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))

        plt.imshow(image, cmap='gray')
        ax.set_title('Label: %d-%d-%s' % (class_id, idx, row['id_code']) )
```

```
CPU times: user 2.78 s, sys: 169 ms, total: 2.95 s
Wall time: 2.82 s
```



For severity level 4, I find it challenging to distinguish two examples, pic(4,1) and pic(4,4). When attempting to zoom in for a closer look at the details (using the real-size image), we can observe certain abnormalities (cotton wool spots or hard exudates?) in those eyes more distinctly (particularly in the lower-right part of the eye). Consequently, IMG_SIZE plays a crucial role in addressing this issue. In the following section, we'll explore a method superior to grayscale conversion.

In [7]:
```
dpi = 80 #inch

# path=f"../input/aptos2019-blindness-detection/train_images/5c7ab966a3ee.png" # notice upper part
path=f"../input/aptos2019-blindness-detection/train_images/cd54d022e37d.png" # lower-right, this still looks not so sev
image = cv2.imread(path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
height, width = image.shape
print(height, width)

SCALE=2
figsize = (width / float(dpi))/SCALE, (height / float(dpi))/SCALE

fig = plt.figure(figsize=figsize)
plt.imshow(image, cmap='gray')
```

```
2136 3216
```

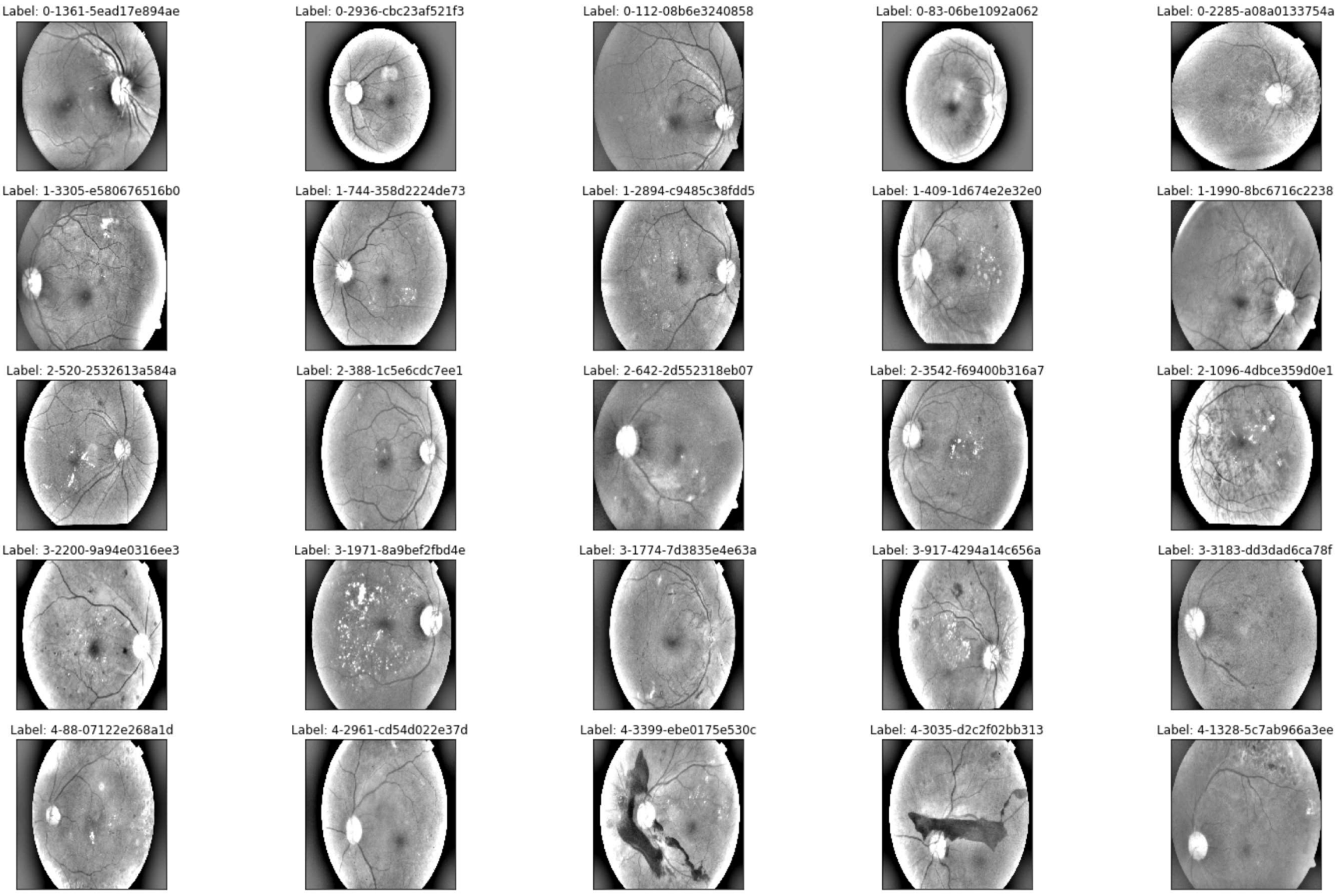
Out[7]:
```
<matplotlib.image.AxesImage at 0x7e5b8c1fa828>
```

Data preprocessing

Improve lighting condition.

In [8]:
```
%%time
fig = plt.figure(figsize=(25, 16))
for class_id in sorted(train_y.unique()):
    for i, (idx, row) in enumerate(df_train.loc[df_train['diagnosis'] == class_id].sample(5, random_state=SEED).iterrow
        ax = fig.add_subplot(5, 5, class_id * 5 + i + 1, xticks=[], yticks=[])
        path=f"../input/aptos2019-blindness-detection/train_images/{row['id_code']}.png"
        image = cv2.imread(path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
#         image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
        image=cv2.addWeighted ( image,4, cv2.GaussianBlur( image , (0,0) , IMG_SIZE/10) ,-4 ,128) # the trick is to add

        plt.imshow(image, cmap='gray')
        ax.set_title('Label: %d-%d-%s' % (class_id, idx, row['id_code']) )
```

```
CPU times: user 18.1 s, sys: 157 ms, total: 18.3 s
Wall time: 7.18 s
```



Further improve by auto-cropping

To eliminate the uninformative black areas present in pic(0,1), pic(0,3), and pic(4,1), we can attempt auto-cropping. Luckily, one method performs impeccably on grayscale images, but none are effective on color images. In this kernel, I've made modifications to adapt the method that works on grayscale images to suit color images.

In [9]:
```
def crop_image1(img,tol=7):
    # img is image data
    # tol  is tolerance

    mask = img>tol
    return img[np.ix_(mask.any(1),mask.any(0))]

def crop_image_from_gray(img,tol=7):
    if img.ndim ==2:
        mask = img>tol
        return img[np.ix_(mask.any(1),mask.any(0))]
    elif img.ndim==3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        mask = gray_img>tol

        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
        if (check_shape == 0): # image is too dark so that we crop out everything,
            return img # return original image
        else:
            img1=img[:,:,0][np.ix_(mask.any(1),mask.any(0))]
            img2=img[:,:,1][np.ix_(mask.any(1),mask.any(0))]
            img3=img[:,:,2][np.ix_(mask.any(1),mask.any(0))]
    #         print(img1.shape,img2.shape,img3.shape)
            img = np.stack([img1,img2,img3],axis=-1)
    #         print(img.shape)
        return img
```

In [10]:
```
# OLD version of image color cropping, use crop_image_from_gray instead
# The above code work only for 1-channel. Here is my simple extension for 3-channels image
def crop_image(img,tol=7):
    if img.ndim ==2:
        mask = img>tol
        return img[np.ix_(mask.any(1),mask.any(0))]
    elif img.ndim==3:
        h,w,_=img.shape
#         print(h,w)
        img1=cv2.resize(crop_image1(img[:,:,0]),(w,h))
        img2=cv2.resize(crop_image1(img[:,:,1]),(w,h))
        img3=cv2.resize(crop_image1(img[:,:,2]),(w,h))

#         print(img1.shape,img2.shape,img3.shape)
        img[:,:,0]=img1
        img[:,:,1]=img2
        img[:,:,2]=img3
        return img

'''all of these do not work'''

def crop_image2(image,threshold=5):
    if len(image.shape) == 3:
```

```
        flatImage = np.max(image, 2)
    else:
        flatImage = image
    assert len(flatImage.shape) == 2

    rows = np.where(np.max(flatImage, 0) > threshold)[0]
    if rows.size:
        cols = np.where(np.max(flatImage, 1) > threshold)[0]
        image = image[cols[0]: cols[-1] + 1, rows[0]: rows[-1] + 1]
    else:
        image = image[:1, :1]

    return image

def crop_image3(image):
    mask = image > 0

    # Coordinates of non-black pixels.
    coords = np.argwhere(mask)

    # Bounding box of non-black pixels.
    x0, y0 = coords.min(axis=0)
    x1, y1 = coords.max(axis=0) + 1   # slices are exclusive at the top

    # Get the contents of the bounding box.
    cropped = image[x0:x1, y0:y1]
    return cropped

def crop_image4(image):
    _,thresh = cv2.threshold(image,1,255,cv2.THRESH_BINARY)
    contours,hierarchy = cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnt = contours[0]
    x,y,w,h = cv2.boundingRect(cnt)
    crop = image[y:y+h,x:x+w]
    return crop
```

Try Cropping the images

I have tested on around 200 images, and the method works great. I think now the eye pictures are very like the moon by the way :)

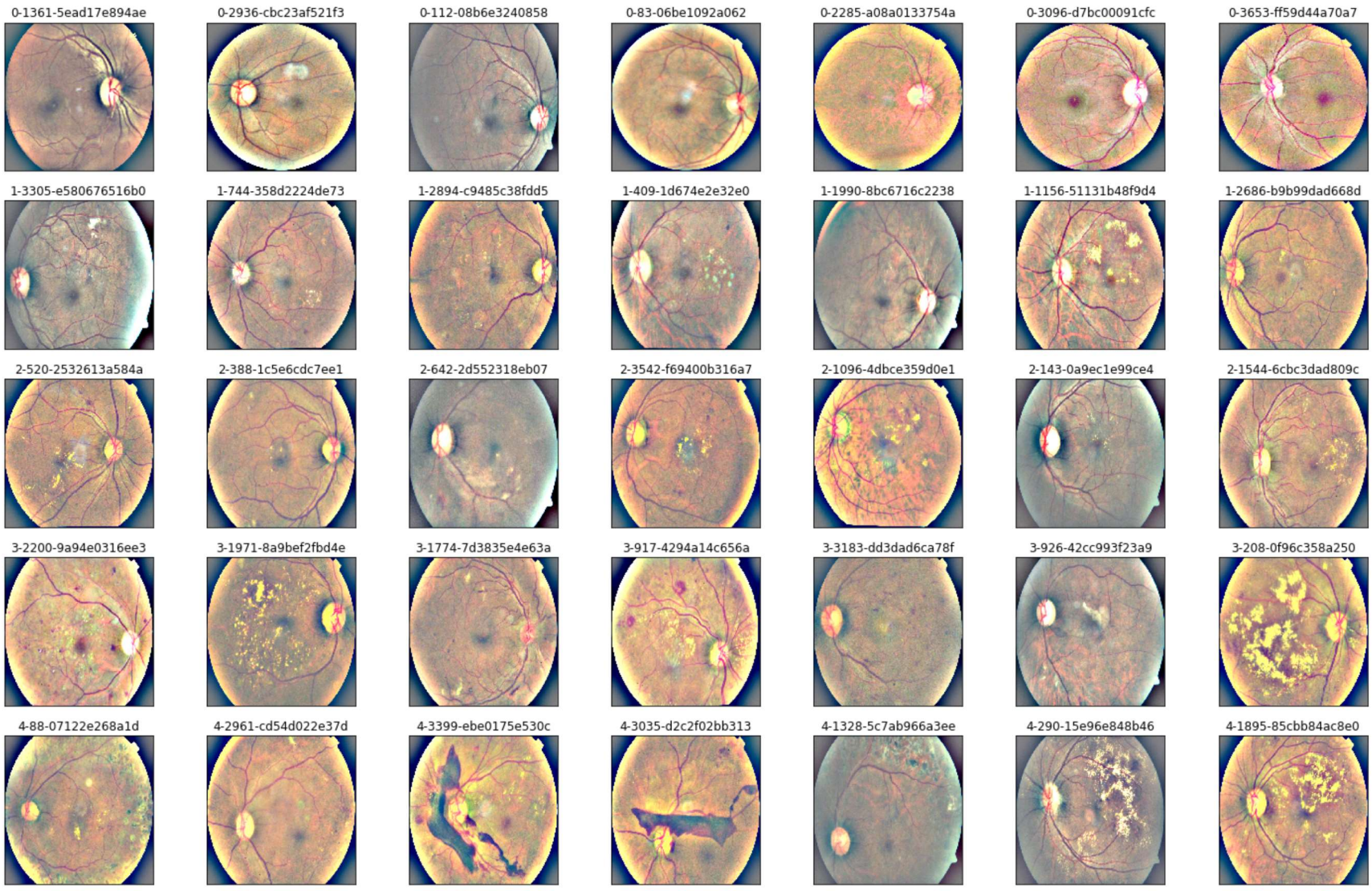
Color Cropping

Below is the cropped version in color. It's worth noting that for the color version, I've utilized the argument sigmaX = 30 with cv2.GaussianBlur, whereas Ben originally used sigmaX = 10, which might yield better performance. Personally, I've found that sigmaX values of 30 or even 50 create strikingly vivid (and at times, intense) yellow moon pictures – just for illustrative purposes.

In [11]:
```
def load_ben_color(path, sigmaX=10):
    image = cv2.imread(path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = crop_image_from_gray(image)
    image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
    image=cv2.addWeighted ( image,4, cv2.GaussianBlur( image , (0,0) , sigmaX) ,-4 ,128)

    return image
```

In [12]:
```
%%time

NUM_SAMP=7
fig = plt.figure(figsize=(25, 16))
for class_id in sorted(train_y.unique()):
    for i, (idx, row) in enumerate(df_train.loc[df_train['diagnosis'] == class_id].sample(NUM_SAMP, random_state=SEED).
        ax = fig.add_subplot(5, NUM_SAMP, class_id * NUM_SAMP + i + 1, xticks=[], yticks=[])
        path=f"../input/aptos2019-blindness-detection/train_images/{row['id_code']}.png"
        image = load_ben_color(path,sigmaX=30)

        plt.imshow(image)
        ax.set_title('%d-%d-%s' % (class_id, idx, row['id_code']) )
```

```
CPU times: user 19.9 s, sys: 303 ms, total: 20.2 s
Wall time: 11.2 s
```



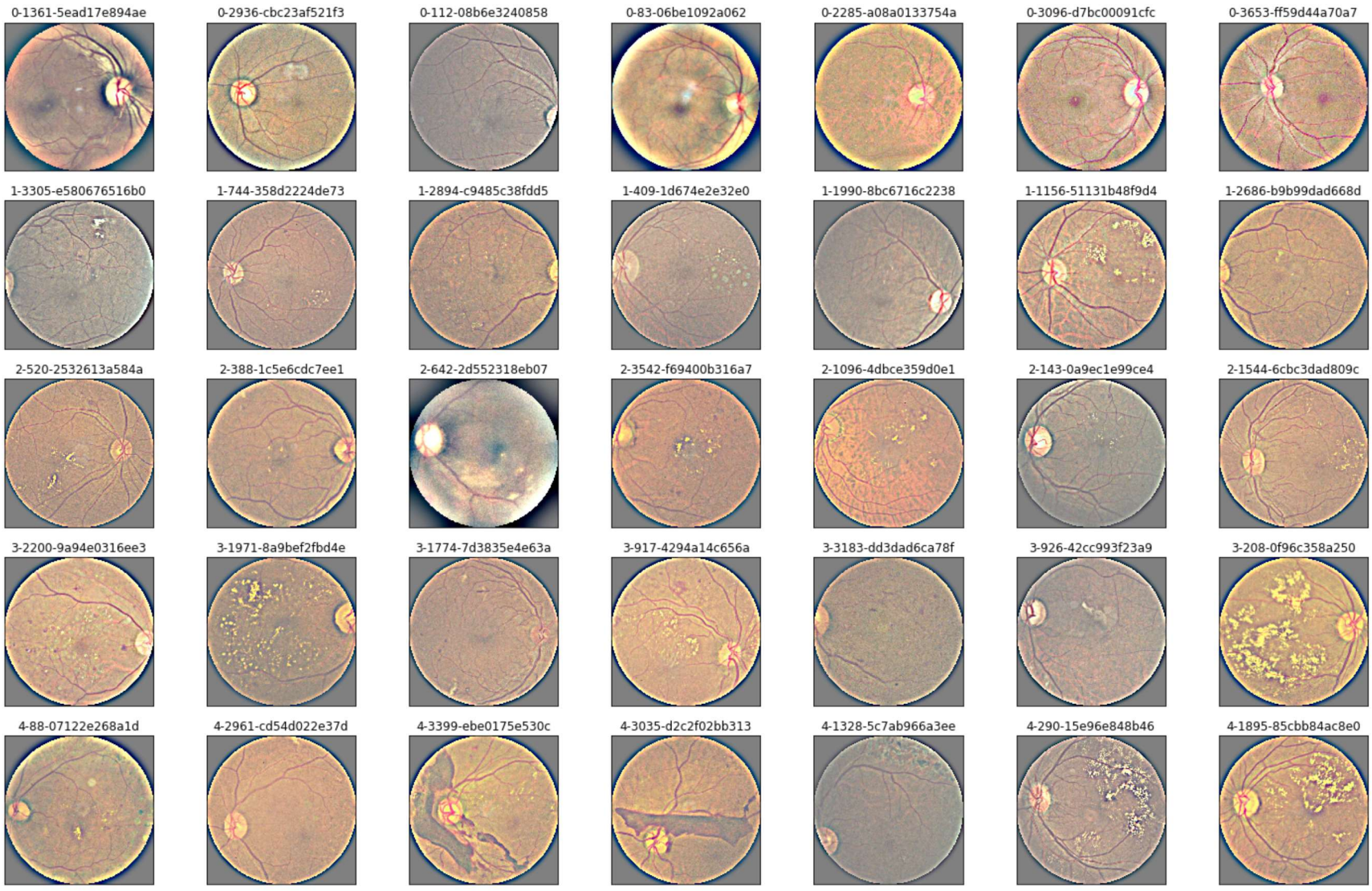
Circle crop to the image

Observe that by using circle crop, some scabs/wools may get loss.

In [13]:
```
def circle_crop(img, sigmaX=10):
    """
    Create circular crop around image centre
    """

    img = cv2.imread(img)
    img = crop_image_from_gray(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    height, width, depth = img.shape

    x = int(width/2)
    y = int(height/2)
    r = np.amin((x,y))

    circle_img = np.zeros((height, width), np.uint8)
    cv2.circle(circle_img, (x,y), int(r), 1, thickness=-1)
    img = cv2.bitwise_and(img, img, mask=circle_img)
    img = crop_image_from_gray(img)
    img=cv2.addWeighted ( img,4, cv2.GaussianBlur( img , (0,0) , sigmaX) ,-4 ,128)
    return img
```

In [14]:
```
%%time
## try circle crop
NUM_SAMP=7
fig = plt.figure(figsize=(25, 16))
for class_id in sorted(train_y.unique()):
    for i, (idx, row) in enumerate(df_train.loc[df_train['diagnosis'] == class_id].sample(NUM_SAMP, random_state=SEED).
        ax = fig.add_subplot(5, NUM_SAMP, class_id * NUM_SAMP + i + 1, xticks=[], yticks=[])
        path=f"../input/aptos2019-blindness-detection/train_images/{row['id_code']}.png"
        image = circle_crop(path,sigmaX=30)

        plt.imshow(image)
        ax.set_title('%d-%d-%s' % (class_id, idx, row['id_code']) )
```

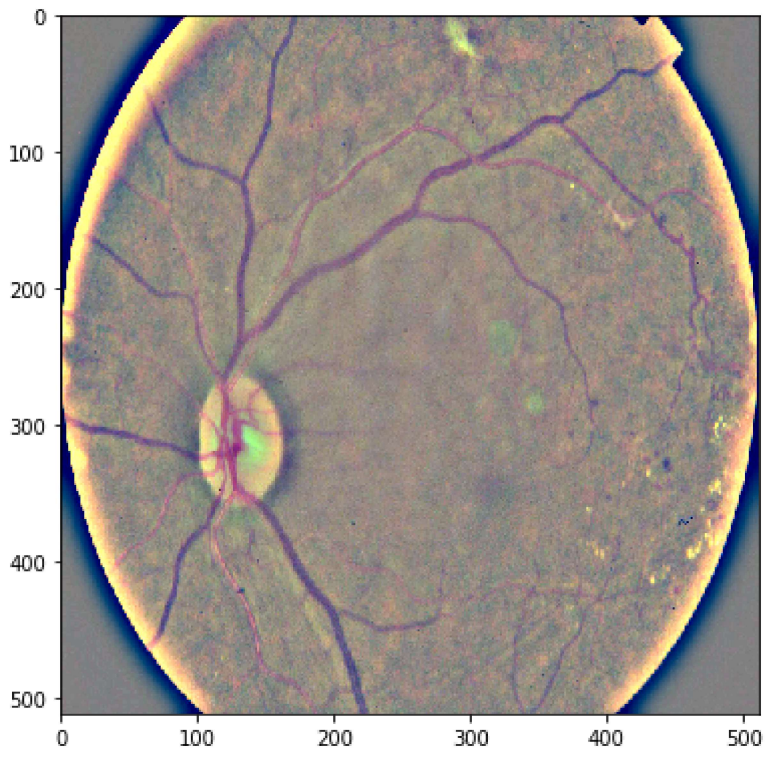
```
CPU times: user 1min 12s, sys: 367 ms, total: 1min 12s
Wall time: 27.5 s
```



We can try plotting a picture (sample train pic(4,1) above) with IMG_SIZE with cropping, now important information is much clearer to see with `sigmaX = 10`

In [15]:
```
dpi = 80 #inch

# path=f"../input/aptos2019-blindness-detection/train_images/5c7ab966a3ee.png" # notice upper part
path=f"../input/aptos2019-blindness-detection/train_images/cd54d022e37d.png" # lower-right, can be class3
image = load_ben_color(path,sigmaX=10)

height, width = IMG_SIZE, IMG_SIZE
print(height, width)

SCALE=1
figsize = (width / float(dpi))/SCALE, (height / float(dpi))/SCALE

fig = plt.figure(figsize=figsize)
plt.imshow(image, cmap='gray')
```

Out[15]:
```
512 512
<matplotlib.image.AxesImage at 0x7e5b8df227f0>
```



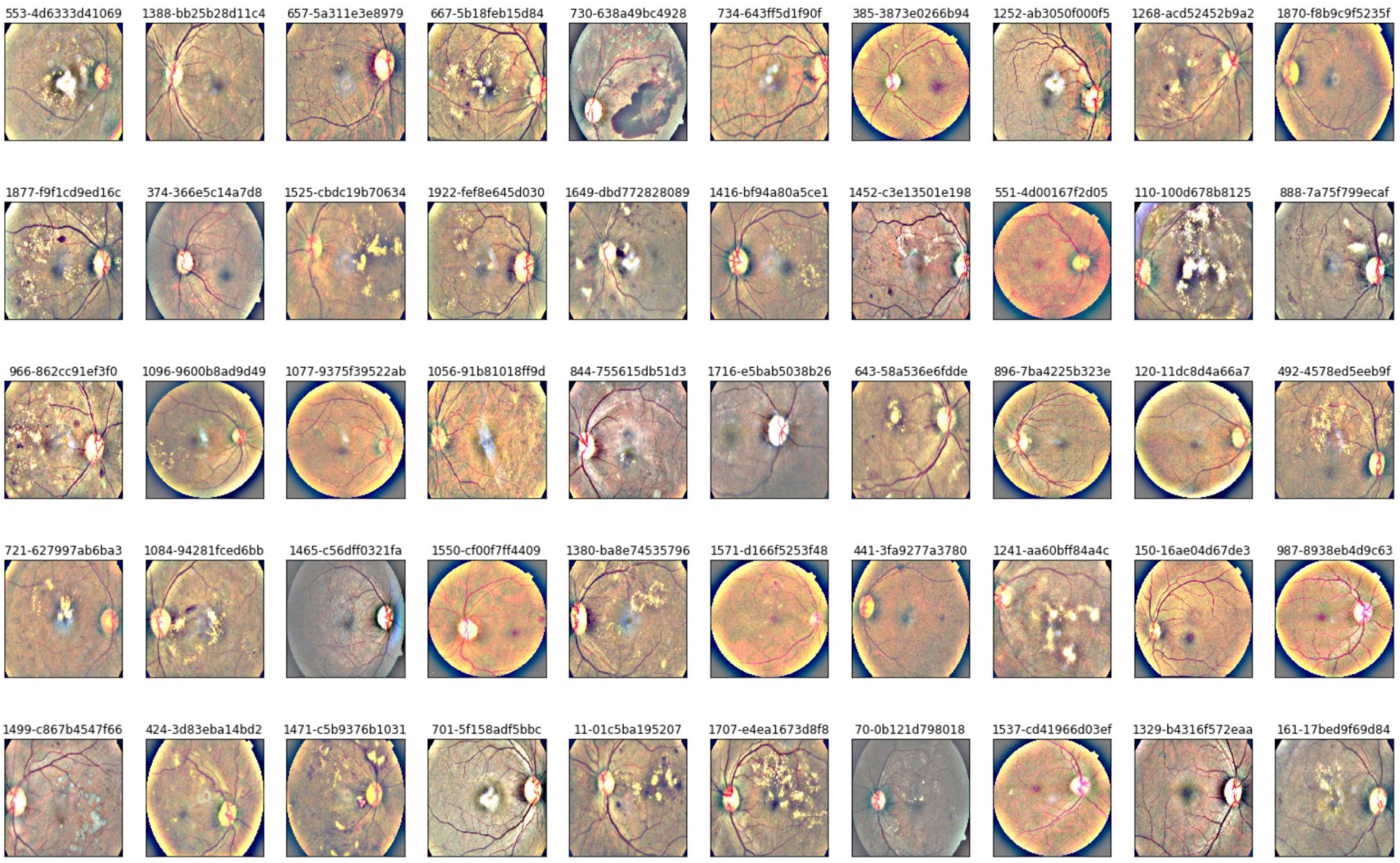
Try the method on Public Test Set

We can also try auto cropping on 50 test data to see that it work fine. Below, we see immediately from this random samples that severed cases, with level >2, are relatively many more compared to the training set.

In [16]:
```
%%time
NUM_SAMP=10
fig = plt.figure(figsize=(25, 16))
for jj in range(5):
    for i, (idx, row) in enumerate(df_test.sample(NUM_SAMP,random_state=SEED+jj).iterrows()):
        ax = fig.add_subplot(5, NUM_SAMP, jj * NUM_SAMP + i + 1, xticks=[], yticks=[])
```

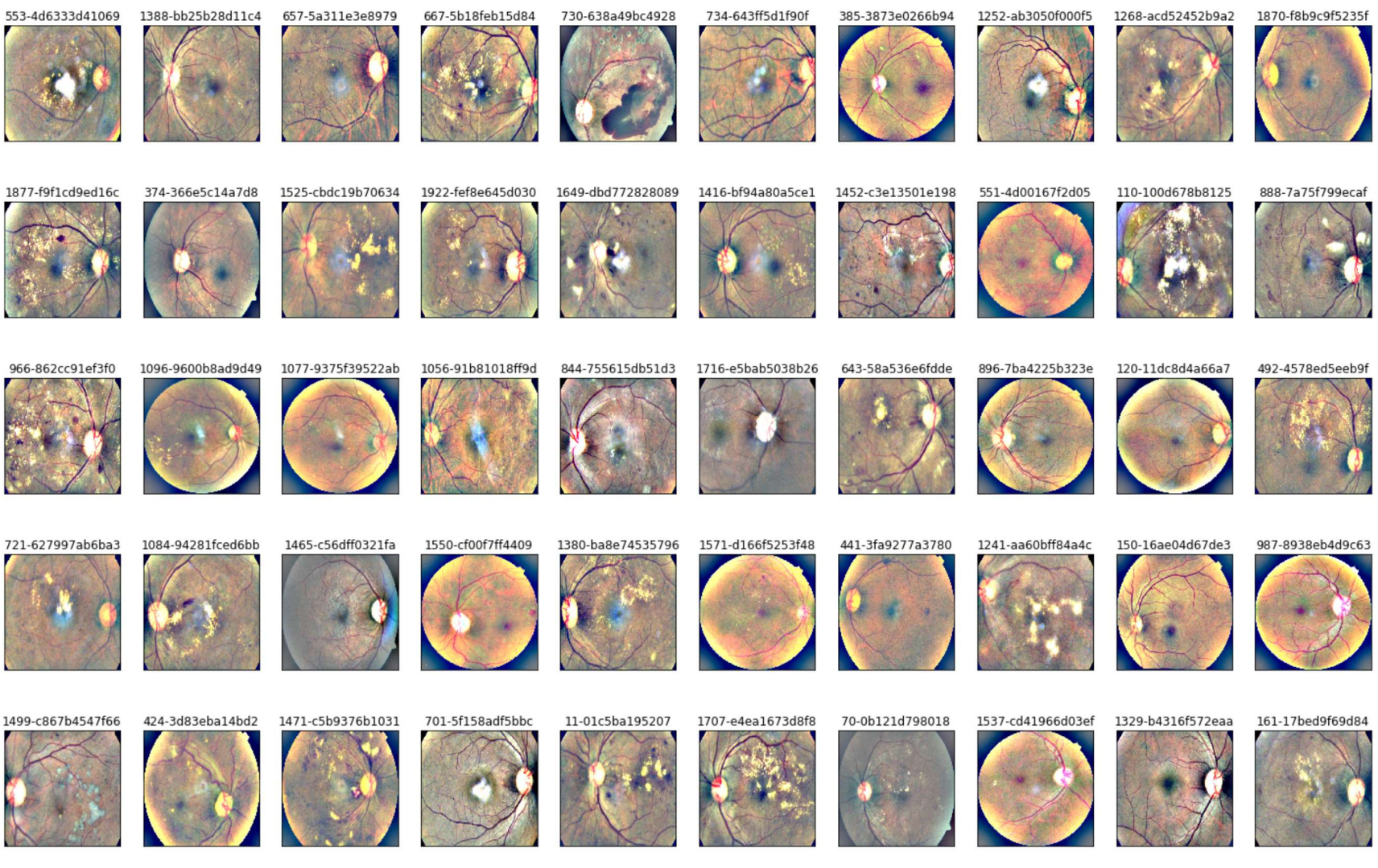
```
        path=f"../input/aptos2019-blindness-detection/test_images/{row['id_code']}.png"
        image = load_ben_color(path,sigmaX=30)

        plt.imshow(image)
        ax.set_title('%d-%s' % (idx, row['id_code']) )
```

```
CPU times: user 21.7 s, sys: 89.6 ms, total: 21.8 s
Wall time: 9.62 s
```



In [17]:
```
%%time
'''Bonus : sigmaX=50'''
NUM_SAMP=10
fig = plt.figure(figsize=(25, 16))
for jj in range(5):
    for i, (idx, row) in enumerate(df_test.sample(NUM_SAMP,random_state=SEED+jj).iterrows()):
        ax = fig.add_subplot(5, NUM_SAMP, jj * NUM_SAMP + i + 1, xticks=[], yticks=[])
        path=f"../input/aptos2019-blindness-detection/test_images/{row['id_code']}.png"
        image = load_ben_color(path,sigmaX=50)

        plt.imshow(image, cmap='gray')
        ax.set_title('%d-%s' % (idx, row['id_code']) )
```

```
CPU times: user 55.7 s, sys: 56 ms, total: 55.7 s
Wall time: 18.3 s
```



In [18]:
```
'''
# This is the old imperfect 'by-channel' color cropping code
# this code can cause different crop among 3 channels

# try cropping color image with the fixed function
```

```
# path=f"../input/aptos2019-blindness-detection/train_images/5c7ab966a3ee.png"
path=f"../input/aptos2019-blindness-detection/train_images/cd54d022e37d.png"
image = cv2.imread(path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = crop_image(image)
# image = crop_image_from_gray(image)
image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
image=cv2.addWeighted ( image,4, cv2.GaussianBlur( image , (0,0) , 10) ,-4 ,128)

height, width = IMG_SIZE, IMG_SIZE
print(height, width)

SCALE=1
figsize = (width / float(dpi))/SCALE, (height / float(dpi))/SCALE

fig = plt.figure(figsize=figsize)
plt.imshow(image)
'''
```

Out[18]:
```
'\n# This is the old imperfect \'by-channel\' color cropping code\n# this code can cause different crop among 3 channel
s\n\n# try cropping color image with the fixed function\n# path=f"../input/aptos2019-blindness-detection/train_images/5
c7ab966a3ee.png"\npath=f"../input/aptos2019-blindness-detection/train_images/cd54d022e37d.png"\nimage = cv2.imread(pat
h)\nimage = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)\nimage = crop_image(image)\n# image = crop_image_from_gray(image)\ni
mage = cv2.resize(image, (IMG_SIZE, IMG_SIZE))\nimage=cv2.addWeighted ( image,4, cv2.GaussianBlur( image , (0,0) , 10)
,-4 ,128)\n\nheight, width = IMG_SIZE, IMG_SIZE\nprint(height, width)\n\nSCALE=1\nfigsize = (width / float(dpi))/SCALE,
(height / float(dpi))/SCALE\n\nfig = plt.figure(figsize=figsize)\nplt.imshow(image)\n'
```

Try the same method to another similar image dataset

In [19]:
```
!ls ../input/diabetic-retinopathy-resized/
```

```
resized_train  resized_train_cropped  trainLabels.csv  trainLabels_cropped.csv
```

In [20]:
```
!ls ../input/diabetic-retinopathy-resized/resized_train/resized_train | head
```

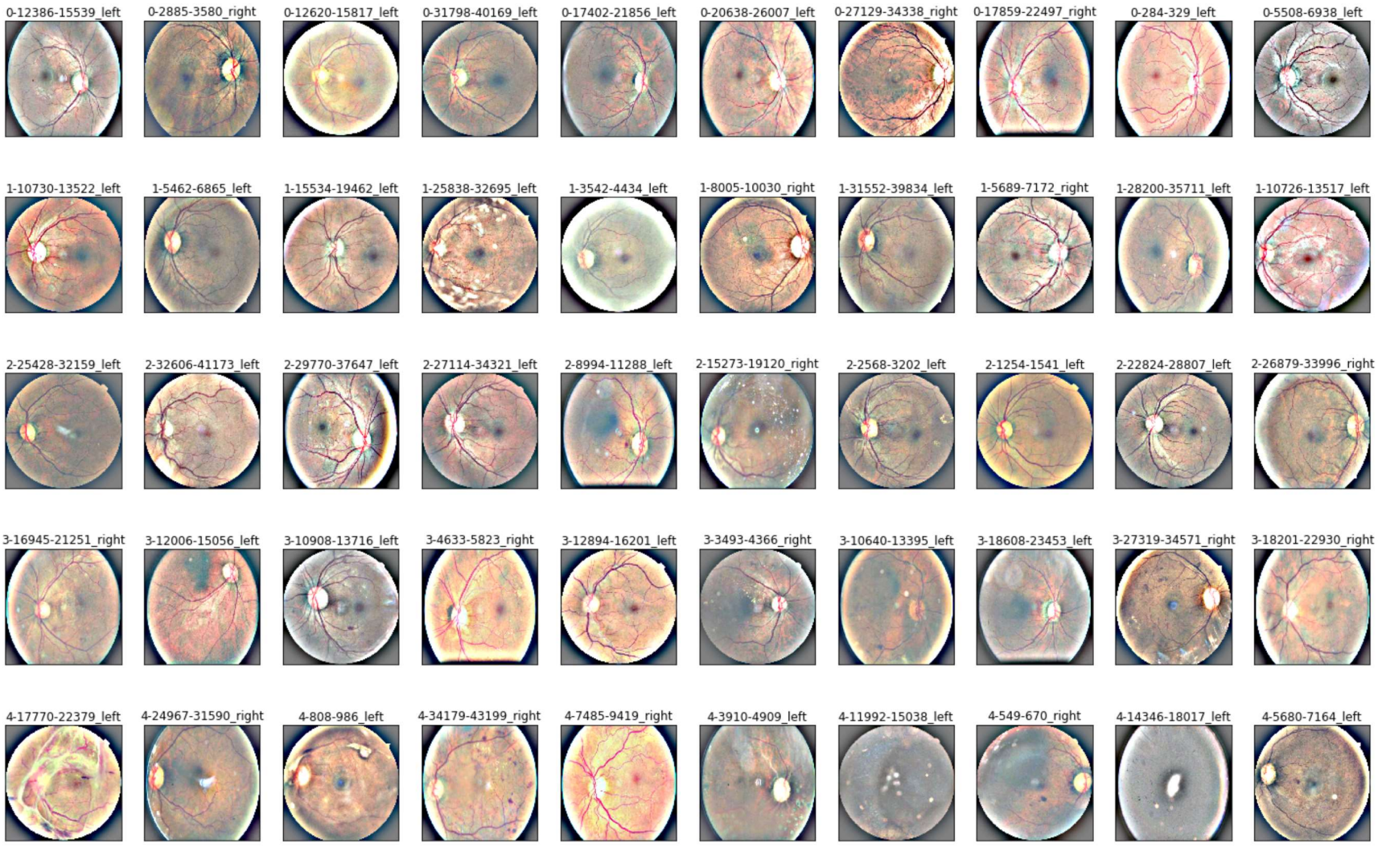
```
10003_left.jpeg
10003_right.jpeg
10007_left.jpeg
10007_right.jpeg
10009_left.jpeg
10009_right.jpeg
1000_left.jpeg
1000_right.jpeg
10010_left.jpeg
10010_right.jpeg
ls: write error: Broken pipe
```

In [21]:
```
df_old = pd.read_csv('../input/diabetic-retinopathy-resized/trainLabels.csv')

df_old.head()
```

Out[21]:

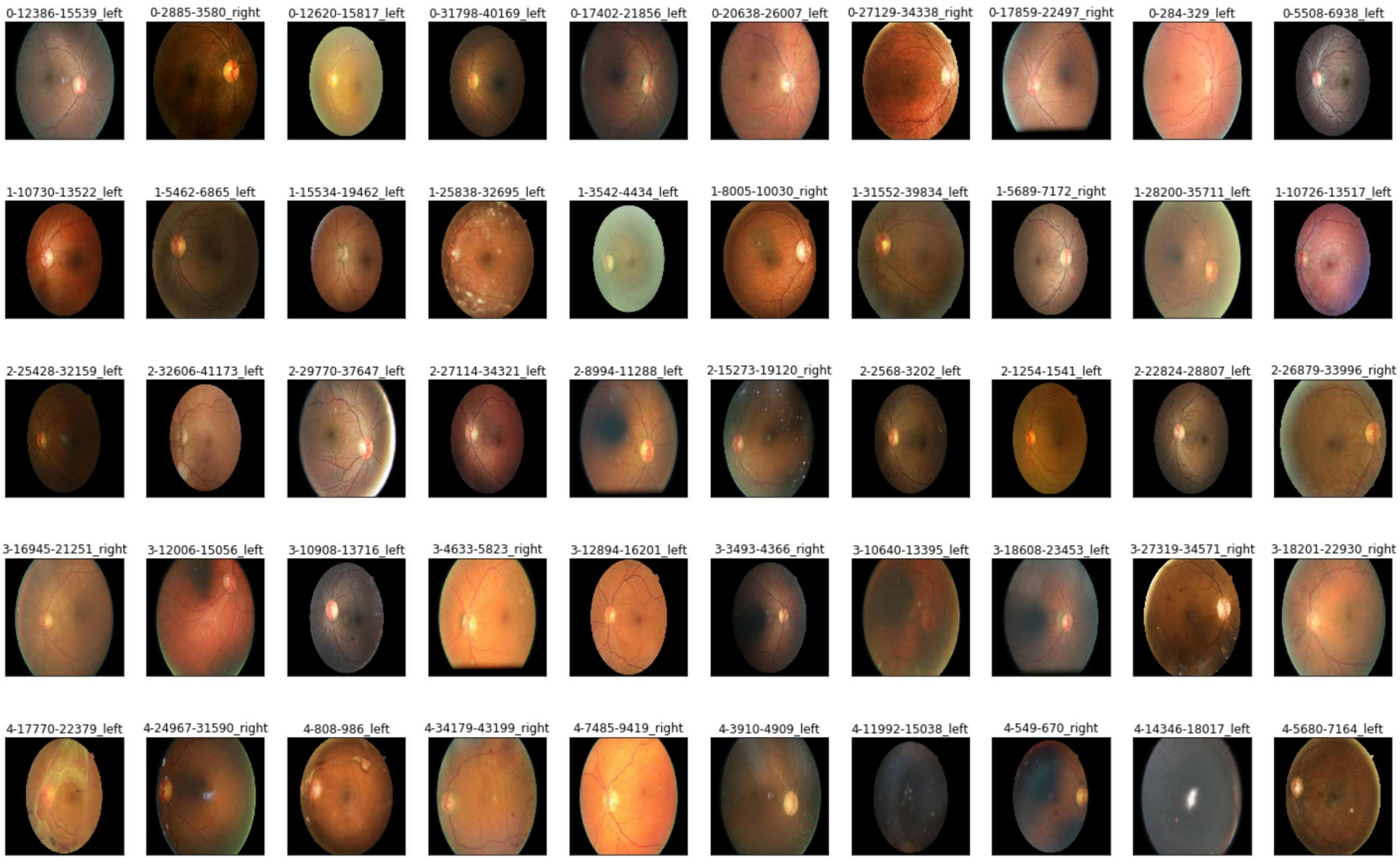
	image	level
0	10_left	0
1	10_right	0
2	13_left	0
3	13_right	0
4	15_left	1

In [22]:
```
NUM_SAMP=10
fig = plt.figure(figsize=(25, 16))
for class_id in sorted(train_y.unique()):
    for i, (idx, row) in enumerate(df_old.loc[df_old['level'] == class_id].sample(NUM_SAMP, random_state=SEED).iterrows
        ax = fig.add_subplot(5, NUM_SAMP, class_id * NUM_SAMP + i + 1, xticks=[], yticks=[])
        path=f"../input/diabetic-retinopathy-resized/resized_train/resized_train/{row['image']}.jpeg"
        image = load_ben_color(path,sigmaX=30)

        plt.imshow(image)
        ax.set_title('%d-%d-%s' % (class_id, idx, row['image']) )
```



Below is the unpreprocess version, just for comparison

In [23]:
```
NUM_SAMP=10
fig = plt.figure(figsize=(25, 16))
for class_id in sorted(train_y.unique()):
    for i, (idx, row) in enumerate(df_old.loc[df_old['level'] == class_id].sample(NUM_SAMP, random_state=SEED).iterrows
        ax = fig.add_subplot(5, NUM_SAMP, class_id * NUM_SAMP + i + 1, xticks=[], yticks=[])
        path=f"../input/diabetic-retinopathy-resized/resized_train/resized_train/{row['image']}.jpeg"
        image = cv2.imread(path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

#         image = crop_image_from_gray(image)
        image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
#         image=cv2.addWeighted ( image,4, cv2.GaussianBlur( image , (0,0) , IMG_SIZE/10) ,-4 ,128)

        plt.imshow(image, cmap='gray')
        ax.set_title('%d-%d-%s' % (class_id, idx, row['image']) )
```



The preprocessing methods appear to work fine. However, it's worth considering that doctors assessing severity levels in past competitions might have had different criteria in mind compared to the doctors at Aravind. Hence, there's a possibility of estimation inconsistency (at least from my perspective; the previous data seems somewhat noisier). The level-4 case (pic(4,1) in the plot above) appears less severe. Alternatively, this might be an instance showcasing excessive blood vessels (refer to Section 1.1).

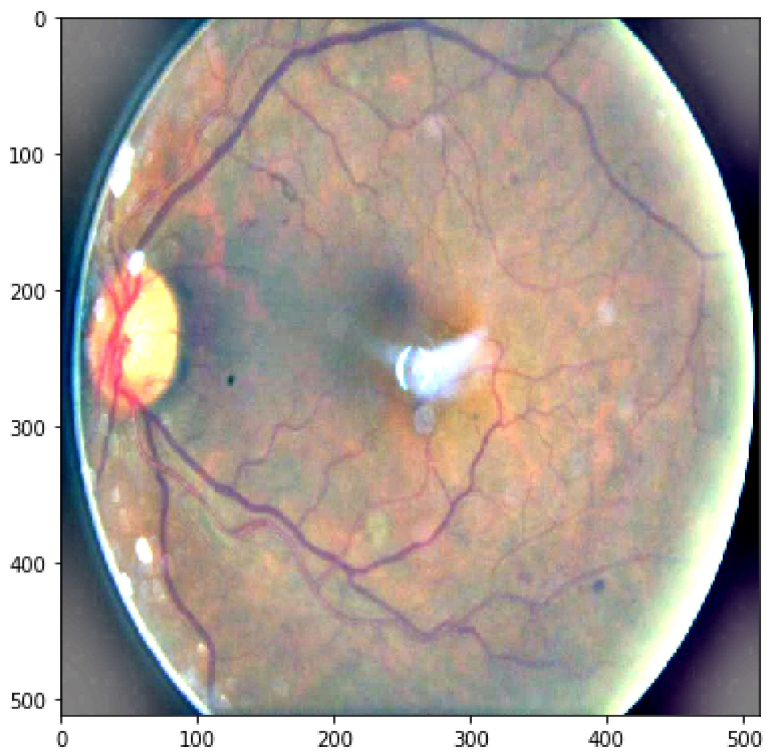
In [24]:
```
dpi = 80 #inch

path=f"../input/diabetic-retinopathy-resized/resized_train/resized_train/31590_right.jpeg" # too many vessels?
```

```
# path=f"../input/diabetic-retinopathy-resized/resized_train/resized_train/18017_left.jpeg" # details are lost
image = load_ben_color(path,sigmaX=30)
# image = cv2.imread(path)
# image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# image = crop_image1(image)
# image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
# image=cv2.addWeighted ( image,4, cv2.GaussianBlur( image , (0,0) , IMG_SIZE/10) ,-4 ,128)

height, width = IMG_SIZE, IMG_SIZE
print(height, width)

SCALE=1
figsize = (width / float(dpi))/SCALE, (height / float(dpi))/SCALE

fig = plt.figure(figsize=figsize)
plt.imshow(image, cmap='gray')
```

```
512 512
```

Out[24]:

```
<matplotlib.image.AxesImage at 0x7e5b8e1ea198>
```



Compare to the .png image.

Some pictures (e.g. pics (4,5-8)) seem to lost details perhaps this is due to .jpeg compression ? I don't think so, but at least we should try to compare with .png.

In [25]:

```
!ls ../input/retinopathy-train-2015/rescaled_train_896/rescaled_train_896/ | head
```

```
10003_left.png
10003_right.png
10007_left.png
10007_right.png
10009_left.png
10009_right.png
1000_left.png
1000_right.png
10010_left.png
10010_right.png
ls: write error: Broken pipe
```

In [26]:

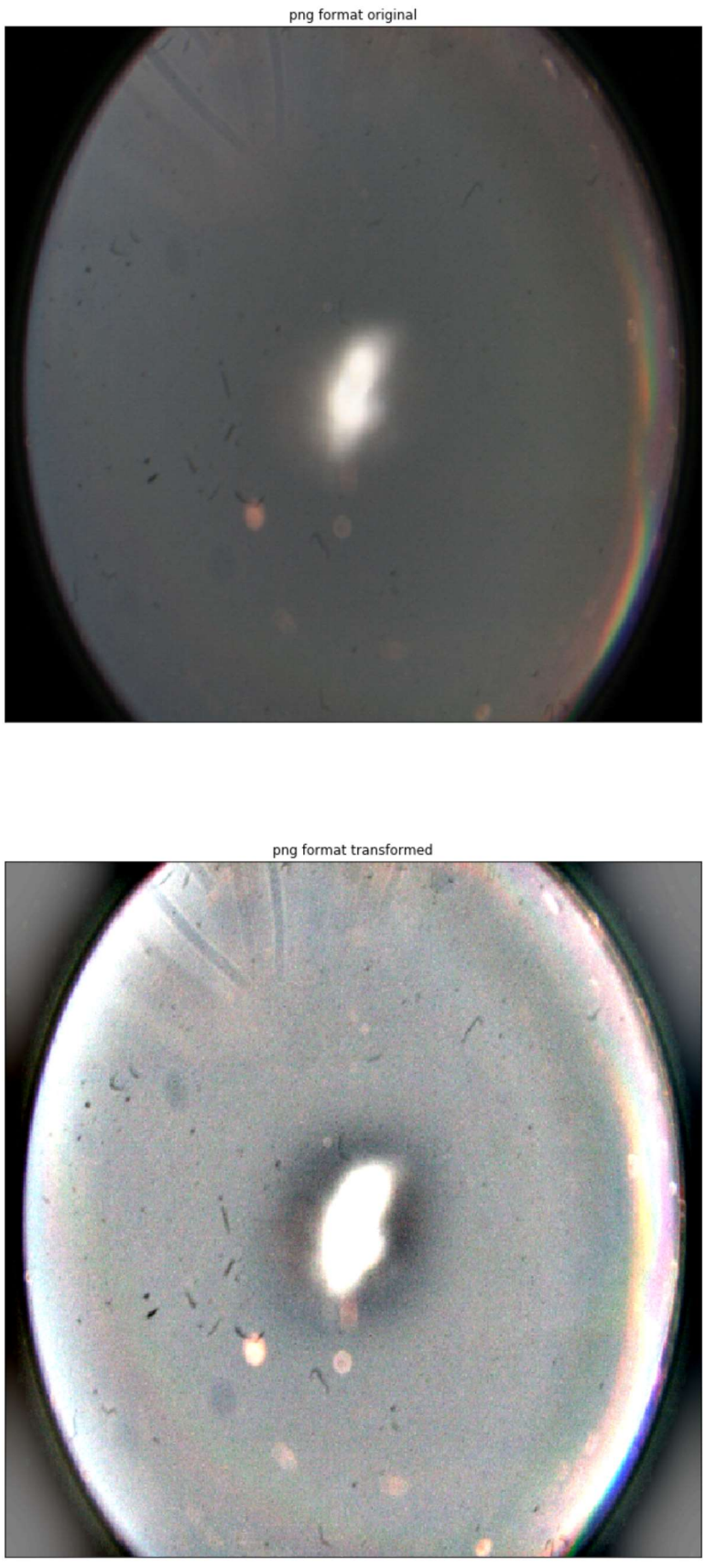
```
dpi = 80 #inch

path_jpg=f"../input/diabetic-retinopathy-resized/resized_train/resized_train/18017_left.jpeg" # too many vessels?
path_png=f"../input/retinopathy-train-2015/rescaled_train_896/rescaled_train_896/18017_left.png" # details are lost
image = cv2.imread(path_png)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = crop_image(image)
image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))

image2 =  cv2.imread(path_jpg)
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
image2 = crop_image(image2)
image2 = cv2.resize(image2, (IMG_SIZE, IMG_SIZE))


height, width = IMG_SIZE, IMG_SIZE
print(height, width)

SCALE=1/4
figsize = (width / float(dpi))/SCALE, (height / float(dpi))/SCALE

fig = plt.figure(figsize=figsize)
ax = fig.add_subplot(2, 2, 1, xticks=[], yticks=[])
ax.set_title('png format original' )
plt.imshow(image, cmap='gray')
ax = fig.add_subplot(2, 2, 2, xticks=[], yticks=[])
ax.set_title('jpg format original' )
plt.imshow(image2, cmap='gray')

image = load_ben_color(path_png,sigmaX=30)
image2 = load_ben_color(path_jpg,sigmaX=30)
```

```
ax = fig.add_subplot(2, 2, 3, xticks=[], yticks=[])
ax.set_title('png format transformed' )
plt.imshow(image, cmap='gray')
ax = fig.add_subplot(2, 2, 4, xticks=[], yticks=[])
ax.set_title('jpg format transformed' )
plt.imshow(image2, cmap='gray')
```

```
512 512
```

Out[26]:

```
<matplotlib.image.AxesImage at 0x7e5b87ead278>
```



The details are really not there no matter what image compression method.