In [1]:
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as pyplot
import matplotlib.pyplot as plt
from IPython.display import HTML
from tensorflow import keras
```

In [2]:
```
import os
path="/kaggle/input/eyes-datasets"
os.chdir(path)
```

In [3]:
```
BATCH_SIZE = 32
length= 224
weidth= 149
CHANNELS=3
EPOCHS=80
```

In [4]:
```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "Eyes_dataset",
    seed=123,
    shuffle=True,
    image_size=(length,weidth),
    batch_size=BATCH_SIZE
)
```

```
Found 5203 files belonging to 3 classes.
```

```
2023-01-06 06:51:24.527870: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:9
37] successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero
2023-01-06 06:51:24.704646: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:9
37] successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero
2023-01-06 06:51:24.705421: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:9
37] successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero
2023-01-06 06:51:24.710617: I tensorflow/core/platform/cpu_feature_guard.cc:142] Thi
s TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to
use the following CPU instructions in performance-critical operations:  AVX2 AVX512F
FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler
flags.
2023-01-06 06:51:24.710908: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:9
37] successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero
2023-01-06 06:51:24.711595: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:9
37] successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero
2023-01-06 06:51:24.712289: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:9
37] successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero
2023-01-06 06:51:27.068553: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:9
37] successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero
2023-01-06 06:51:27.069391: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:9
37] successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero
2023-01-06 06:51:27.070045: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:9
37] successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero
2023-01-06 06:51:27.070627: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510]
Created device /job:localhost/replica:0/task:0/device:GPU:0 with 15401 MB memory:  -
> device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capabilit
y: 6.0
```

In [5]:

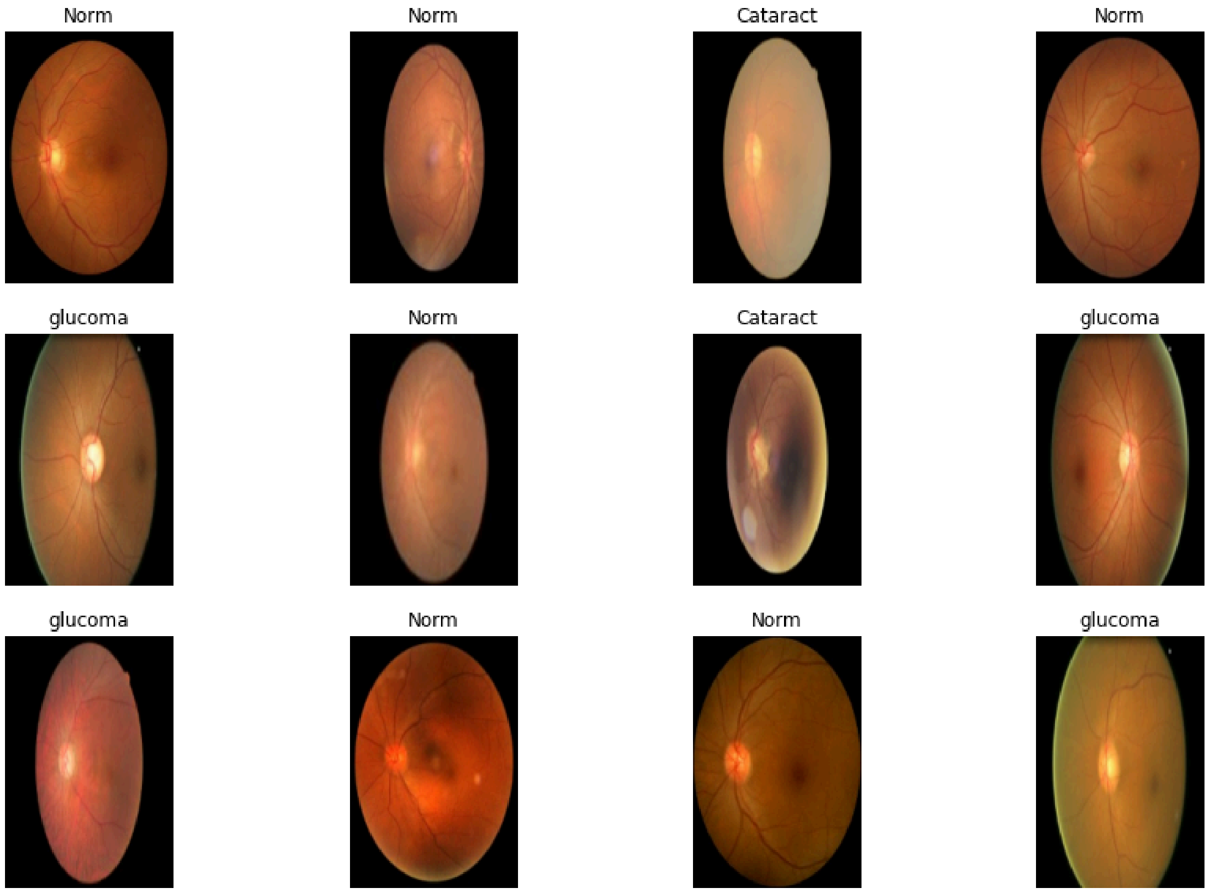
```
class_names = dataset.class_names
class_names
```

Out[5]:

```
['Cataract', 'Norm', 'glucoma']
```

In [8]:

```
plt.figure(figsize=(15, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



In [6]:
```
len(dataset)
```

Out[6]: 163

In [7]:
```
train_size = 0.7
len(dataset)*train_size
```

Out[7]: 114.1

In [8]:
```
test_ds = dataset.skip(114)
len(test_ds)
```

Out[8]: 49

In [9]:
```
def get_dataset_partitions_tf(ds, train_split=0.7, val_split=0.10, test_split=0.20,
    #assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
```

```
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

In [10]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

In [11]:
```
len(val_ds)
```

Out[11]: 16

In [13]:
```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(length,weidth),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

In [14]:
```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

In [ ]:
```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [14]:
```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

In [15]:
```
model = keras.models.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape = [224, 149,3]),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(120, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(150, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(164, (2, 2), activation='relu'),
    keras.layers.Flatten(),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(3, activation ='softmax')
])
```

In [16]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 222, 147, 32)      896       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 73, 32)       0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 71, 64)       18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 35, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 52, 33, 64)        36928     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 26, 16, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 14, 120)       69240     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 7, 120)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 5, 150)        162150    
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 2, 150)         0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 1, 164)         98564     
_________________________________________________________________
flatten (Flatten)            (None, 656)               0         
_________________________________________________________________
dense (Dense)                (None, 150)               98550     
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 453       
=================================================================
Total params: 485,277
Trainable params: 485,277
Non-trainable params: 0
_________________________________________________________________
```

In [17]:

```python
for i in range(len(model.layers)):

# check for convolutional layer
    if 'conv' not in model.layers[i].name:
        continue
# get filter weights
    filters, biases = model.layers[i].get_weights()
    print("layer number",i,model.layers[i].name, filters.shape)
```

```
layer number 0 conv2d (3, 3, 3, 32)
layer number 2 conv2d_1 (3, 3, 32, 64)
layer number 4 conv2d_2 (3, 3, 64, 64)
layer number 6 conv2d_3 (3, 3, 64, 120)
layer number 8 conv2d_4 (3, 3, 120, 150)
layer number 10 conv2d_5 (2, 2, 150, 164)
```

In [58]:

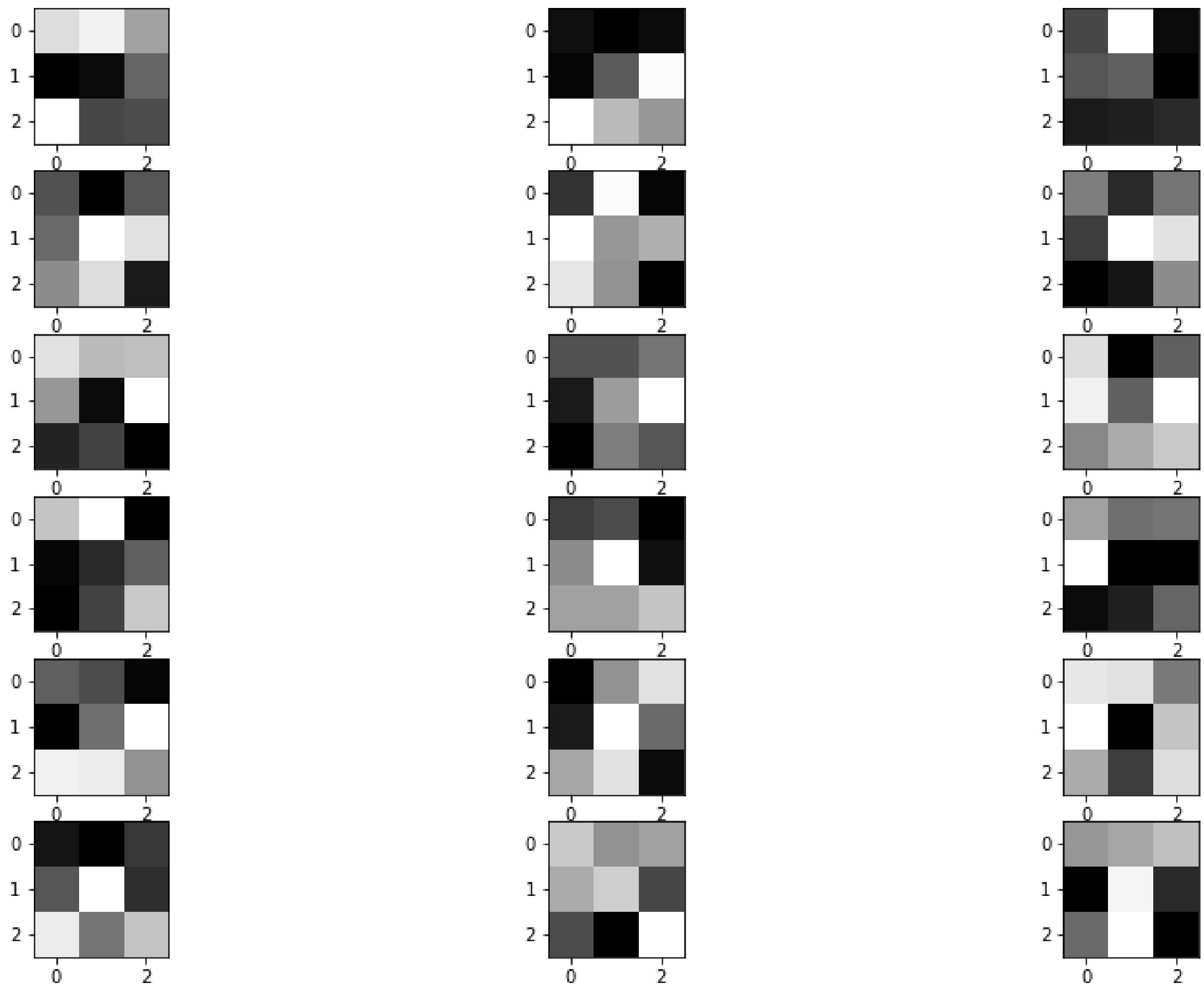
```python
# retrieve weights from the second hidden layer
filters , bias = model.layers[2].get_weights()
```

In [59]:
```
# normalize filter values to 0-1 so we can visualize them
f_min, f_max = filters.min(), filters.max()
filters = (filters - f_min) / (f_max - f_min)
```

In [60]:
```
from keras.models import Model
#from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from numpy import expand_dims
```

In [20]:
```
model = Model(inputs=model.inputs , outputs=model.layers[2].output)
```

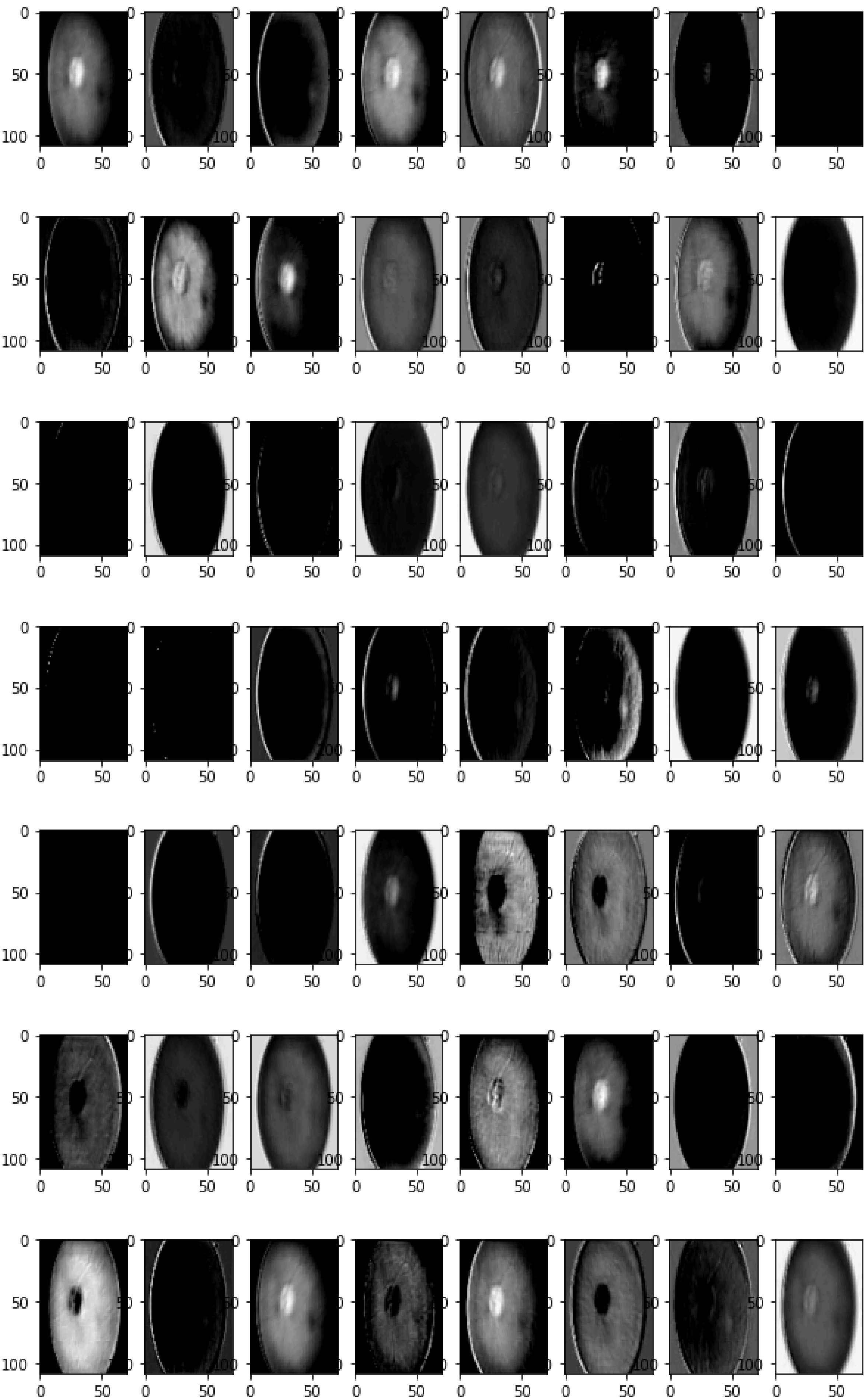
In [21]:
```
n_filters =6
ix=1
fig = pyplot.figure(figsize=(15,10))
for i in range(n_filters):
    # get the filters
    f = filters[:,:,:,i]
    for j in range(3):
        # subplot for 6 filters and 3 channels
        pyplot.subplot(n_filters,3,ix)
        pyplot.imshow(f[:,:,j] ,cmap='gray')
        ix+=1
#plot the filters
pyplot.show()
```

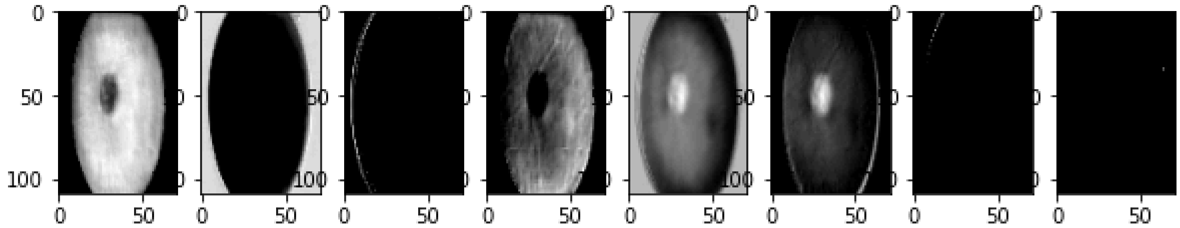


In [ ]:
```
# redefine model to output right after the first hidden layer
model = Model(inputs=model.inputs, outputs=model.layers[2].output)
```

In [31]:
```
import numpy as np
image = tf.keras.utils.load_img('/kaggle/input/eyes-datasets/Eyes_dataset/Norm/003
# convert the image to an array
image = img_to_array(image)
# expand dimensions so that it represents a single 'sample'
image = expand_dims(image, axis=0)

image = preprocess_input(image)
```

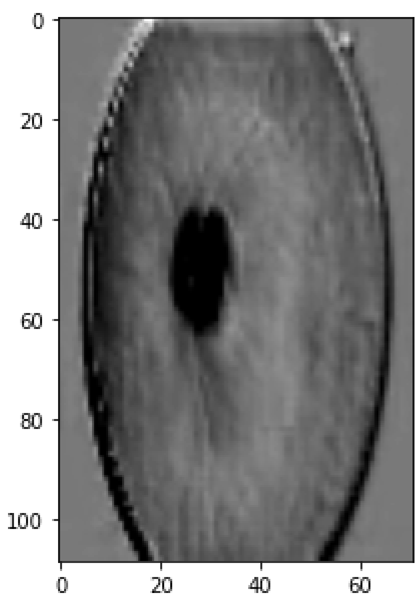
In [32]:
```
#calculating features_map
features = model.predict(image)

fig = pyplot.figure(figsize=(10,20))
for i in range(1,features.shape[3]+1):

    pyplot.subplot(8,8,i)
    pyplot.imshow(features[0,:,:,i-1] , cmap='gray')

pyplot.show()
```





```
In [45]: #calculating features_map
         features = model.predict(image)
         fig = pyplot.figure(figsize=(5,5))

         pyplot.imshow(features[0,:,:,i-27] , cmap='gray')
```

```
Out[45]: <matplotlib.image.AxesImage at 0x7f57581e9350>
```




```
In [18]: model.compile(
             optimizer='adam',
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
             metrics=['accuracy']
         )
```

```
In [19]: history=model.fit(
             train_ds,
             batch_size=BATCH_SIZE,
             validation_data=val_ds,
              epochs= 90
         )
```

```
Epoch 1/90
2023-01-06 06:52:57.286701: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.
cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
2023-01-06 06:53:06.750049: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loade
d cuDNN version 8005
```

```
114/114 [==============================] - 27s 94ms/step - loss: 0.9146 - accuracy:
0.6512 - val_loss: 0.6336 - val_accuracy: 0.7074
Epoch 2/90
114/114 [==============================] - 11s 66ms/step - loss: 0.6001 - accuracy:
0.7373 - val_loss: 0.6503 - val_accuracy: 0.6875
Epoch 3/90
114/114 [==============================] - 11s 65ms/step - loss: 0.5685 - accuracy:
0.7323 - val_loss: 0.5103 - val_accuracy: 0.8027
Epoch 4/90
114/114 [==============================] - 11s 67ms/step - loss: 0.5118 - accuracy:
0.7549 - val_loss: 0.5157 - val_accuracy: 0.7635
Epoch 5/90
114/114 [==============================] - 10s 66ms/step - loss: 0.4902 - accuracy:
0.7747 - val_loss: 0.5109 - val_accuracy: 0.7715
Epoch 6/90
114/114 [==============================] - 11s 67ms/step - loss: 0.4738 - accuracy:
0.7769 - val_loss: 0.4700 - val_accuracy: 0.7734
Epoch 7/90
114/114 [==============================] - 11s 71ms/step - loss: 0.4841 - accuracy:
0.7703 - val_loss: 0.4971 - val_accuracy: 0.7227
Epoch 8/90
114/114 [==============================] - 11s 69ms/step - loss: 0.4547 - accuracy:
0.7873 - val_loss: 0.4116 - val_accuracy: 0.8156
Epoch 9/90
114/114 [==============================] - 11s 72ms/step - loss: 0.4401 - accuracy:
0.7961 - val_loss: 0.3871 - val_accuracy: 0.8184
Epoch 10/90
114/114 [==============================] - 11s 70ms/step - loss: 0.4357 - accuracy:
0.7963 - val_loss: 0.4021 - val_accuracy: 0.8242
Epoch 11/90
114/114 [==============================] - 11s 69ms/step - loss: 0.4395 - accuracy:
0.7950 - val_loss: 0.4206 - val_accuracy: 0.7969
Epoch 12/90
114/114 [==============================] - 10s 63ms/step - loss: 0.4124 - accuracy:
0.8048 - val_loss: 0.4216 - val_accuracy: 0.8277
Epoch 13/90
114/114 [==============================] - 11s 68ms/step - loss: 0.4078 - accuracy:
0.8117 - val_loss: 0.3837 - val_accuracy: 0.8223
Epoch 14/90
114/114 [==============================] - 10s 67ms/step - loss: 0.3905 - accuracy:
0.8217 - val_loss: 0.3501 - val_accuracy: 0.8555
Epoch 15/90
114/114 [==============================] - 11s 68ms/step - loss: 0.3883 - accuracy:
0.8176 - val_loss: 0.3378 - val_accuracy: 0.8105
Epoch 16/90
114/114 [==============================] - 11s 67ms/step - loss: 0.3826 - accuracy:
0.8195 - val_loss: 0.3343 - val_accuracy: 0.8301
Epoch 17/90
114/114 [==============================] - 10s 69ms/step - loss: 0.3691 - accuracy:
0.8309 - val_loss: 0.3468 - val_accuracy: 0.8301
Epoch 18/90
114/114 [==============================] - 11s 67ms/step - loss: 0.3885 - accuracy:
0.8232 - val_loss: 0.3182 - val_accuracy: 0.8594
Epoch 19/90
114/114 [==============================] - 11s 69ms/step - loss: 0.3549 - accuracy:
0.8405 - val_loss: 0.3626 - val_accuracy: 0.8574
```

```
Epoch 20/90
114/114 [==============================] - 10s 67ms/step - loss: 0.3597 - accuracy:
0.8292 - val_loss: 0.3384 - val_accuracy: 0.8477
Epoch 21/90
114/114 [==============================] - 11s 70ms/step - loss: 0.3356 - accuracy:
0.8451 - val_loss: 0.3070 - val_accuracy: 0.8496
Epoch 22/90
114/114 [==============================] - 10s 66ms/step - loss: 0.3348 - accuracy:
0.8459 - val_loss: 0.3291 - val_accuracy: 0.8555
Epoch 23/90
114/114 [==============================] - 11s 70ms/step - loss: 0.3339 - accuracy:
0.8487 - val_loss: 0.3166 - val_accuracy: 0.8496
Epoch 24/90
114/114 [==============================] - 10s 65ms/step - loss: 0.3174 - accuracy:
0.8514 - val_loss: 0.2924 - val_accuracy: 0.8657
Epoch 25/90
114/114 [==============================] - 10s 66ms/step - loss: 0.3207 - accuracy:
0.8479 - val_loss: 0.2910 - val_accuracy: 0.8730
Epoch 26/90
114/114 [==============================] - 11s 72ms/step - loss: 0.3298 - accuracy:
0.8473 - val_loss: 0.2813 - val_accuracy: 0.8672
Epoch 27/90
114/114 [==============================] - 10s 67ms/step - loss: 0.3105 - accuracy:
0.8599 - val_loss: 0.2924 - val_accuracy: 0.8535
Epoch 28/90
114/114 [==============================] - 11s 72ms/step - loss: 0.3164 - accuracy:
0.8550 - val_loss: 0.2931 - val_accuracy: 0.8594
Epoch 29/90
114/114 [==============================] - 10s 65ms/step - loss: 0.2926 - accuracy:
0.8594 - val_loss: 0.2509 - val_accuracy: 0.8809
Epoch 30/90
114/114 [==============================] - 10s 70ms/step - loss: 0.2809 - accuracy:
0.8641 - val_loss: 0.2368 - val_accuracy: 0.8848
Epoch 31/90
114/114 [==============================] - 10s 64ms/step - loss: 0.2727 - accuracy:
0.8702 - val_loss: 0.2751 - val_accuracy: 0.8730
Epoch 32/90
114/114 [==============================] - 10s 63ms/step - loss: 0.2743 - accuracy:
0.8676 - val_loss: 0.2562 - val_accuracy: 0.8737
Epoch 33/90
114/114 [==============================] - 10s 64ms/step - loss: 0.2326 - accuracy:
0.8868 - val_loss: 0.2333 - val_accuracy: 0.8887
Epoch 34/90
114/114 [==============================] - 11s 72ms/step - loss: 0.2473 - accuracy:
0.8847 - val_loss: 0.2869 - val_accuracy: 0.8613
Epoch 35/90
114/114 [==============================] - 10s 64ms/step - loss: 0.2402 - accuracy:
0.8834 - val_loss: 0.1990 - val_accuracy: 0.9062
Epoch 36/90
114/114 [==============================] - 10s 66ms/step - loss: 0.2474 - accuracy:
0.8787 - val_loss: 0.2087 - val_accuracy: 0.8945
Epoch 37/90
114/114 [==============================] - 11s 68ms/step - loss: 0.2603 - accuracy:
0.8698 - val_loss: 0.1933 - val_accuracy: 0.9023
Epoch 38/90
114/114 [==============================] - 11s 68ms/step - loss: 0.2365 - accuracy:
```

```
0.8842 - val_loss: 0.2527 - val_accuracy: 0.8809
Epoch 39/90
114/114 [==============================] - 11s 67ms/step - loss: 0.2202 - accuracy:
0.8875 - val_loss: 0.1795 - val_accuracy: 0.9023
Epoch 40/90
114/114 [==============================] - 12s 81ms/step - loss: 0.2244 - accuracy:
0.8891 - val_loss: 0.2018 - val_accuracy: 0.8958
Epoch 41/90
114/114 [==============================] - 11s 64ms/step - loss: 0.2352 - accuracy:
0.8871 - val_loss: 0.1703 - val_accuracy: 0.9141
Epoch 42/90
114/114 [==============================] - 11s 64ms/step - loss: 0.1984 - accuracy:
0.8979 - val_loss: 0.1773 - val_accuracy: 0.9018
Epoch 43/90
114/114 [==============================] - 10s 66ms/step - loss: 0.2003 - accuracy:
0.8974 - val_loss: 0.1932 - val_accuracy: 0.8906
Epoch 44/90
114/114 [==============================] - 11s 70ms/step - loss: 0.1940 - accuracy:
0.8999 - val_loss: 0.2042 - val_accuracy: 0.8926
Epoch 45/90
114/114 [==============================] - 10s 68ms/step - loss: 0.2297 - accuracy:
0.8839 - val_loss: 0.2467 - val_accuracy: 0.8438
Epoch 46/90
114/114 [==============================] - 11s 65ms/step - loss: 0.2076 - accuracy:
0.8931 - val_loss: 0.2184 - val_accuracy: 0.9023
Epoch 47/90
114/114 [==============================] - 10s 66ms/step - loss: 0.1996 - accuracy:
0.9019 - val_loss: 0.2018 - val_accuracy: 0.8984
Epoch 48/90
114/114 [==============================] - 10s 65ms/step - loss: 0.2205 - accuracy:
0.8917 - val_loss: 0.1999 - val_accuracy: 0.8711
Epoch 49/90
114/114 [==============================] - 10s 68ms/step - loss: 0.1836 - accuracy:
0.8996 - val_loss: 0.2123 - val_accuracy: 0.8828
Epoch 50/90
114/114 [==============================] - 10s 65ms/step - loss: 0.2066 - accuracy:
0.8983 - val_loss: 0.1993 - val_accuracy: 0.8965
Epoch 51/90
114/114 [==============================] - 10s 69ms/step - loss: 0.1780 - accuracy:
0.9012 - val_loss: 0.1900 - val_accuracy: 0.9023
Epoch 52/90
114/114 [==============================] - 10s 63ms/step - loss: 0.1800 - accuracy:
0.9027 - val_loss: 0.1877 - val_accuracy: 0.8965
Epoch 53/90
114/114 [==============================] - 10s 67ms/step - loss: 0.1898 - accuracy:
0.9052 - val_loss: 0.2440 - val_accuracy: 0.8867
Epoch 54/90
114/114 [==============================] - 10s 62ms/step - loss: 0.1928 - accuracy:
0.9010 - val_loss: 0.2006 - val_accuracy: 0.8926
Epoch 55/90
114/114 [==============================] - 10s 67ms/step - loss: 0.1804 - accuracy:
0.9037 - val_loss: 0.1904 - val_accuracy: 0.8828
Epoch 56/90
114/114 [==============================] - 10s 68ms/step - loss: 0.2038 - accuracy:
0.8983 - val_loss: 0.1642 - val_accuracy: 0.9121
Epoch 57/90
```

```
114/114 [==============================] - 10s 64ms/step - loss: 0.1683 - accuracy:
0.9117 - val_loss: 0.1548 - val_accuracy: 0.9121
Epoch 58/90
114/114 [==============================] - 11s 69ms/step - loss: 0.1803 - accuracy:
0.9054 - val_loss: 0.2029 - val_accuracy: 0.8906
Epoch 59/90
114/114 [==============================] - 11s 69ms/step - loss: 0.2004 - accuracy:
0.8977 - val_loss: 0.1770 - val_accuracy: 0.9180
Epoch 60/90
114/114 [==============================] - 11s 71ms/step - loss: 0.1643 - accuracy:
0.9106 - val_loss: 0.1754 - val_accuracy: 0.8965
Epoch 61/90
114/114 [==============================] - 10s 67ms/step - loss: 0.1642 - accuracy:
0.9128 - val_loss: 0.1295 - val_accuracy: 0.9180
Epoch 62/90
114/114 [==============================] - 10s 64ms/step - loss: 0.1805 - accuracy:
0.9037 - val_loss: 0.1825 - val_accuracy: 0.9141
Epoch 63/90
114/114 [==============================] - 11s 69ms/step - loss: 0.1637 - accuracy:
0.9078 - val_loss: 0.1791 - val_accuracy: 0.8984
Epoch 64/90
114/114 [==============================] - 11s 71ms/step - loss: 0.1781 - accuracy:
0.9043 - val_loss: 0.1859 - val_accuracy: 0.8878
Epoch 65/90
114/114 [==============================] - 10s 64ms/step - loss: 0.1865 - accuracy:
0.8988 - val_loss: 0.1397 - val_accuracy: 0.9336
Epoch 66/90
114/114 [==============================] - 10s 66ms/step - loss: 0.1629 - accuracy:
0.9120 - val_loss: 0.1882 - val_accuracy: 0.8998
Epoch 67/90
114/114 [==============================] - 11s 67ms/step - loss: 0.1511 - accuracy:
0.9202 - val_loss: 0.1571 - val_accuracy: 0.9043
Epoch 68/90
114/114 [==============================] - 10s 65ms/step - loss: 0.1479 - accuracy:
0.9189 - val_loss: 0.1553 - val_accuracy: 0.9078
Epoch 69/90
114/114 [==============================] - 11s 69ms/step - loss: 0.1679 - accuracy:
0.9052 - val_loss: 0.1613 - val_accuracy: 0.8898
Epoch 70/90
114/114 [==============================] - 10s 67ms/step - loss: 0.1828 - accuracy:
0.9010 - val_loss: 0.1908 - val_accuracy: 0.8906
Epoch 71/90
114/114 [==============================] - 10s 70ms/step - loss: 0.2139 - accuracy:
0.8927 - val_loss: 0.1811 - val_accuracy: 0.8965
Epoch 72/90
114/114 [==============================] - 10s 66ms/step - loss: 0.1631 - accuracy:
0.9133 - val_loss: 0.1392 - val_accuracy: 0.9199
Epoch 73/90
114/114 [==============================] - 11s 71ms/step - loss: 0.1928 - accuracy:
0.9015 - val_loss: 0.1925 - val_accuracy: 0.8945
Epoch 74/90
114/114 [==============================] - 10s 66ms/step - loss: 0.2000 - accuracy:
0.8988 - val_loss: 0.2275 - val_accuracy: 0.8848
Epoch 75/90
114/114 [==============================] - 11s 71ms/step - loss: 0.2053 - accuracy:
0.8952 - val_loss: 0.1482 - val_accuracy: 0.9062
```

```
Epoch 76/90
114/114 [==============================] - 10s 62ms/step - loss: 0.1695 - accuracy:
0.9054 - val_loss: 0.1643 - val_accuracy: 0.8945
Epoch 77/90
114/114 [==============================] - 10s 63ms/step - loss: 0.1795 - accuracy:
0.9062 - val_loss: 0.1718 - val_accuracy: 0.9238
Epoch 78/90
114/114 [==============================] - 11s 67ms/step - loss: 0.1708 - accuracy:
0.9046 - val_loss: 0.2060 - val_accuracy: 0.8887
Epoch 79/90
114/114 [==============================] - 10s 66ms/step - loss: 0.1582 - accuracy:
0.9109 - val_loss: 0.1595 - val_accuracy: 0.9121
Epoch 80/90
114/114 [==============================] - 11s 71ms/step - loss: 0.1514 - accuracy:
0.9093 - val_loss: 0.1594 - val_accuracy: 0.9082
Epoch 81/90
114/114 [==============================] - 10s 68ms/step - loss: 0.1633 - accuracy:
0.9098 - val_loss: 0.1273 - val_accuracy: 0.9339
Epoch 82/90
114/114 [==============================] - 11s 67ms/step - loss: 0.1379 - accuracy:
0.9147 - val_loss: 0.1688 - val_accuracy: 0.9004
Epoch 83/90
114/114 [==============================] - 10s 68ms/step - loss: 0.1388 - accuracy:
0.9155 - val_loss: 0.1292 - val_accuracy: 0.9178
Epoch 84/90
114/114 [==============================] - 10s 65ms/step - loss: 0.1375 - accuracy:
0.9128 - val_loss: 0.1393 - val_accuracy: 0.9062
Epoch 85/90
114/114 [==============================] - 10s 65ms/step - loss: 0.1389 - accuracy:
0.9104 - val_loss: 0.1164 - val_accuracy: 0.9316
Epoch 86/90
114/114 [==============================] - 10s 64ms/step - loss: 0.1336 - accuracy:
0.9216 - val_loss: 0.1318 - val_accuracy: 0.9004
Epoch 87/90
114/114 [==============================] - 10s 65ms/step - loss: 0.1325 - accuracy:
0.9134 - val_loss: 0.1403 - val_accuracy: 0.8958
Epoch 88/90
114/114 [==============================] - 10s 67ms/step - loss: 0.1598 - accuracy:
0.9060 - val_loss: 0.2446 - val_accuracy: 0.8878
Epoch 89/90
114/114 [==============================] - 10s 64ms/step - loss: 0.2273 - accuracy:
0.8880 - val_loss: 0.1648 - val_accuracy: 0.9180
Epoch 90/90
114/114 [==============================] - 11s 71ms/step - loss: 0.1829 - accuracy:
0.9016 - val_loss: 0.1593 - val_accuracy: 0.9121
```

In [74]:

```
import os
path="/kaggle/working/"
os.chdir(path)
```

In [75]:

```
model.save('Eyesprediction.h5')  # creates a HDF5 file 'my_model.h5'
```

In [78]:

```
scores = model.evaluate(test_ds)
```

```
33/33 [==============================] - 4s 8ms/step - loss: 0.1647 - accuracy: 0.91
47
```

In [79]:
```python
history
```

Out[79]:
```
<keras.callbacks.History at 0x7fb65a2520d0>
```

In [80]:
```python
history.params
```

Out[80]:
```
{'verbose': 1, 'epochs': 80, 'steps': 114}
```

In [26]:
```python
history.history.keys()
```

Out[26]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [77]:
```python
type(history.history['loss'])
```

Out[77]:
```
list
```

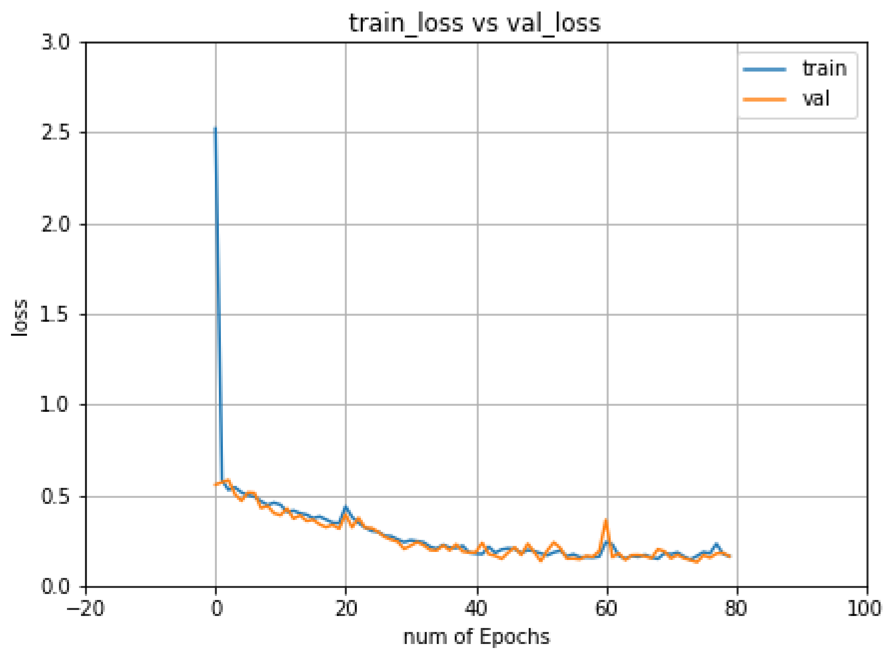
In [78]:
```python
len(history.history['loss'])
```

Out[78]:
```
100
```

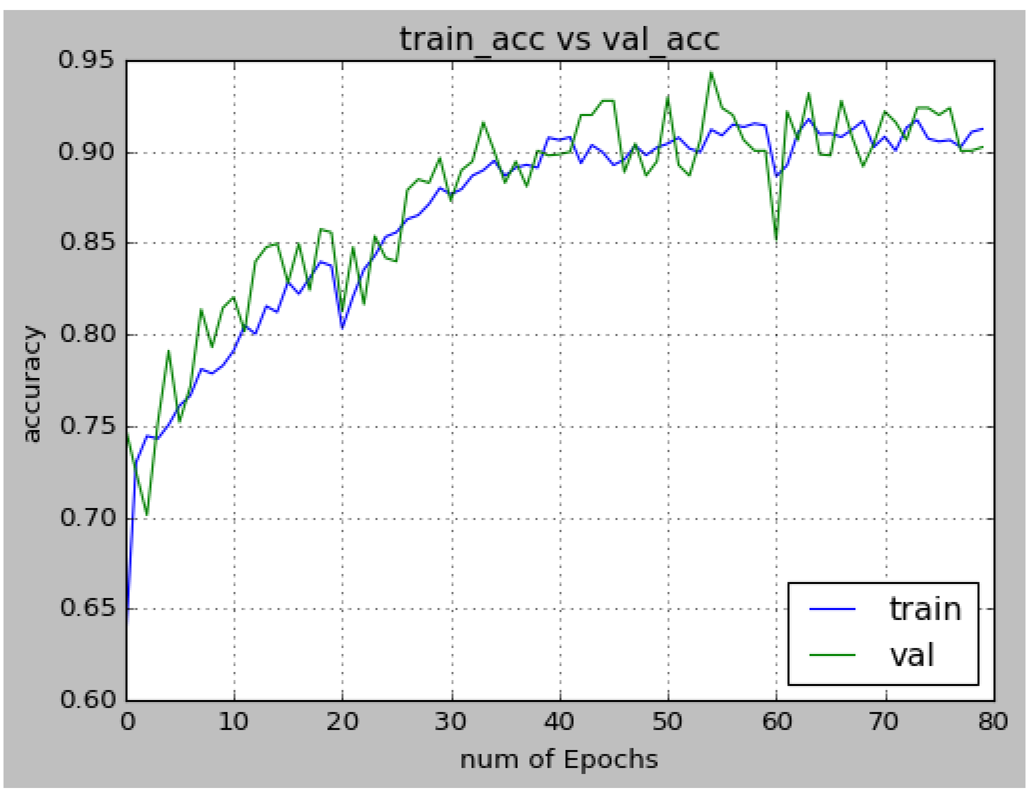
In [26]:
```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

In [27]:
```python
train_loss=history.history['loss']
val_loss=history.history['val_loss']
train_acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
xc=range(80)
```

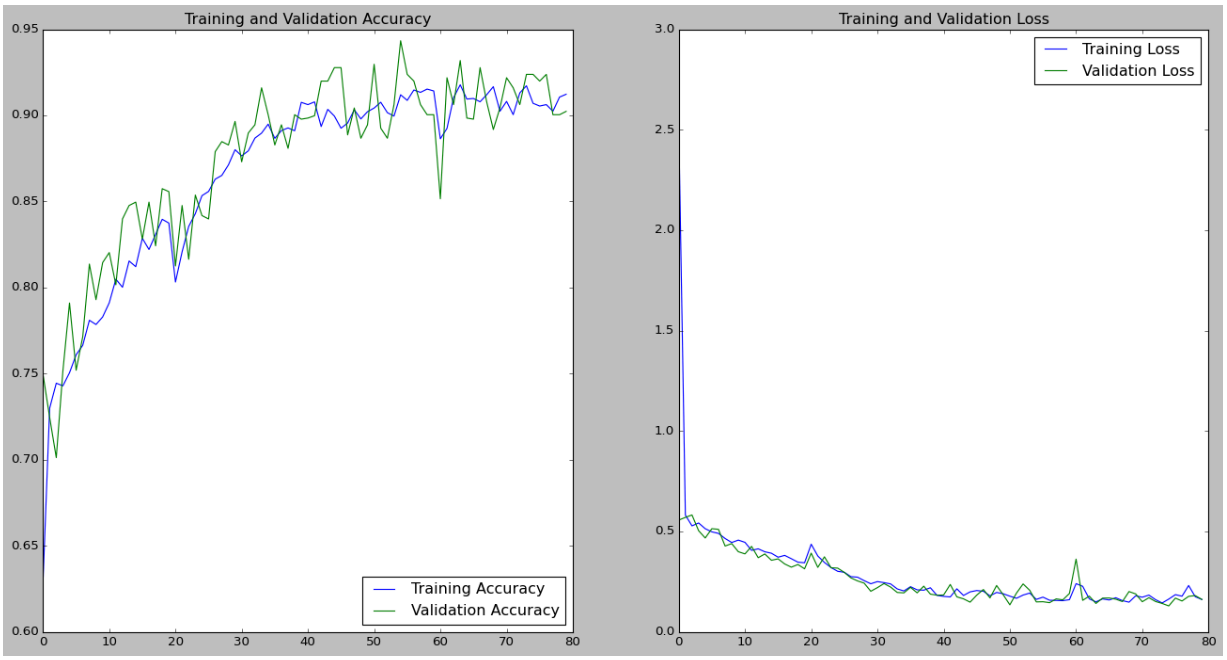
In [31]:
```python
plt.figure(1,figsize=(7,5))
plt.plot(xc,train_loss)
plt.plot(xc,val_loss)
plt.xlabel('num of Epochs')
plt.ylabel('loss')
plt.title('train_loss vs val_loss')
plt.grid(True)
plt.legend(['train','val'])
plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])
```



In [32]:
```
plt.figure(2,figsize=(7,5))
plt.plot(xc,train_acc)
plt.plot(xc,val_acc)
plt.xlabel('num of Epochs')
plt.ylabel('accuracy')
plt.title('train_acc vs val_acc')
plt.grid(True)
plt.legend(['train','val'],loc=4)
#print plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])
```

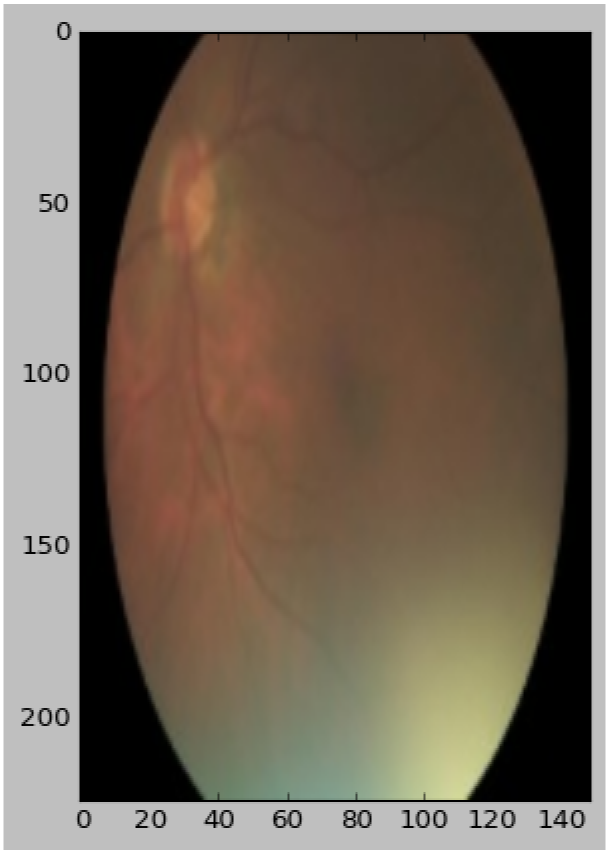


In [54]:
```
plt.figure(figsize=(20, 10))
plt.subplot(1, 2, 1)
plt.plot(range(80), acc, label='Training Accuracy')
plt.plot(range(80), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(80), loss, label='Training Loss')
plt.plot(range(80), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



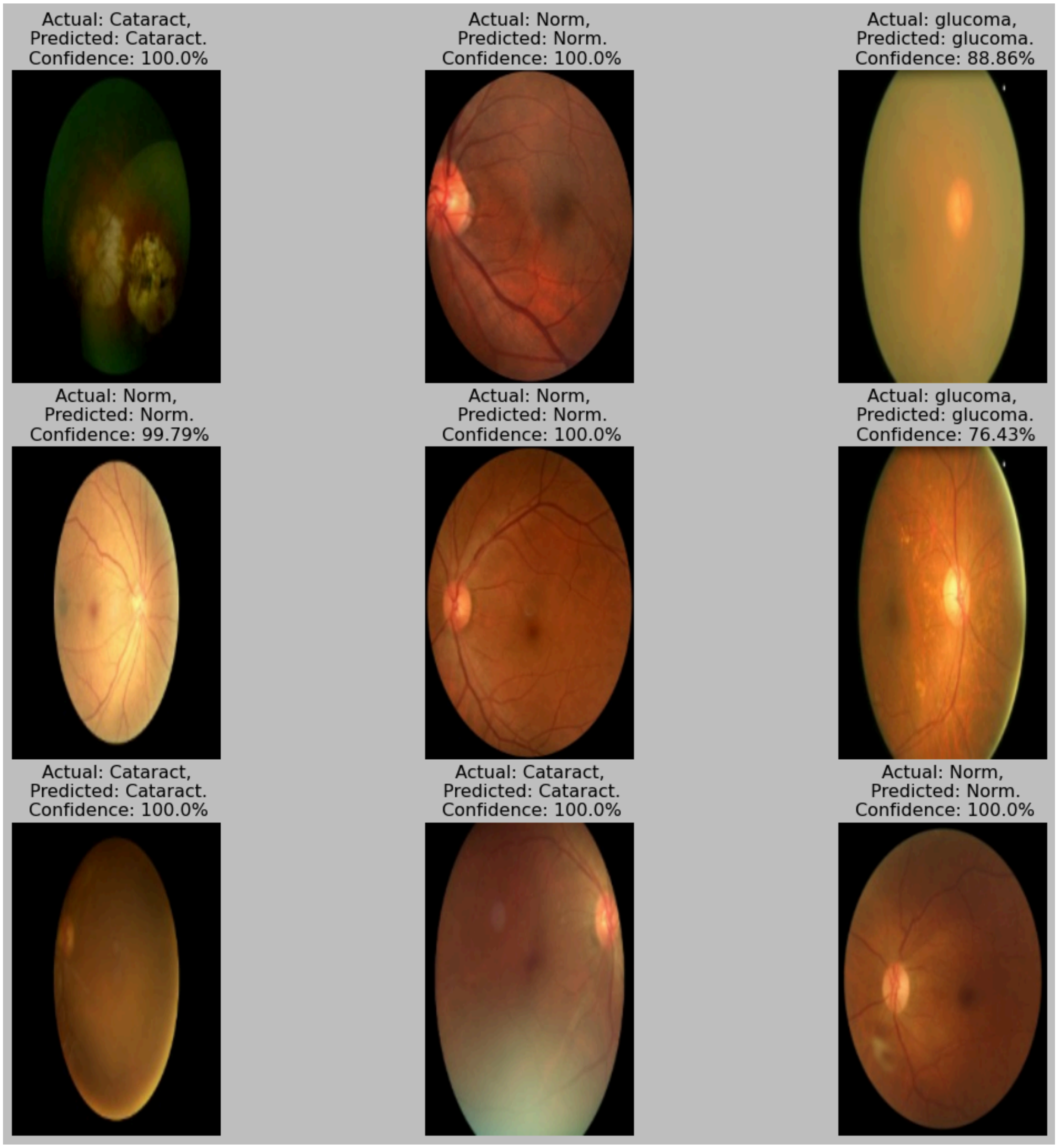
In [52]:
```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[1].numpy().astype('uint8')
    first_label = labels_batch[1].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[1])])
```

```
first image to predict
actual label: Cataract
predicted label: Cataract
```



In [53]:
```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

In [58]:
```
plt.figure(figsize=(17, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confi

        plt.axis("off")
```


Actual: Cataract,
Predicted: Cataract.
Confidence: 100.0%
Actual: Norm,
Predicted: Norm.
Confidence: 100.0%
Actual: glucoma,
Predicted: glucoma.
Confidence: 88.86%
Actual: Norm,
Predicted: Norm.
Confidence: 99.79%
Actual: Norm,
Predicted: Norm.
Confidence: 100.0%
Actual: glucoma,
Predicted: glucoma.
Confidence: 76.43%
Actual: Cataract,
Predicted: Cataract.
Confidence: 100.0%
Actual: Cataract,
Predicted: Cataract.
Confidence: 100.0%
Actual: Norm,
Predicted: Norm.
Confidence: 100.0%