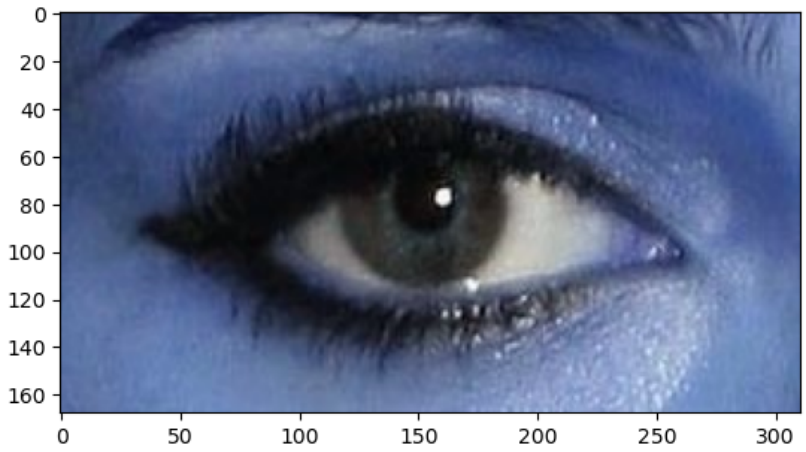
Eye_Cataract detection using CNN algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.preprocessing import image_dataset_from_directory

DATADIR = "/Users/kartiksolanki/Desktop/eye_cataract/image/train"
CATEGORIES = ["normal", "cataract"]

for category in CATEGORIES:
    path = os.path.join(DATADIR, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img))
        plt.imshow(img_array, cmap="gray")
        plt.show()
        break
    break
```




```
print(img_array.shape)
```

(168, 311, 3)

```
def create_data(path):
    data = []
    for category in CATEGORIES:
        path = os.path.join(DATADIR, category)
```

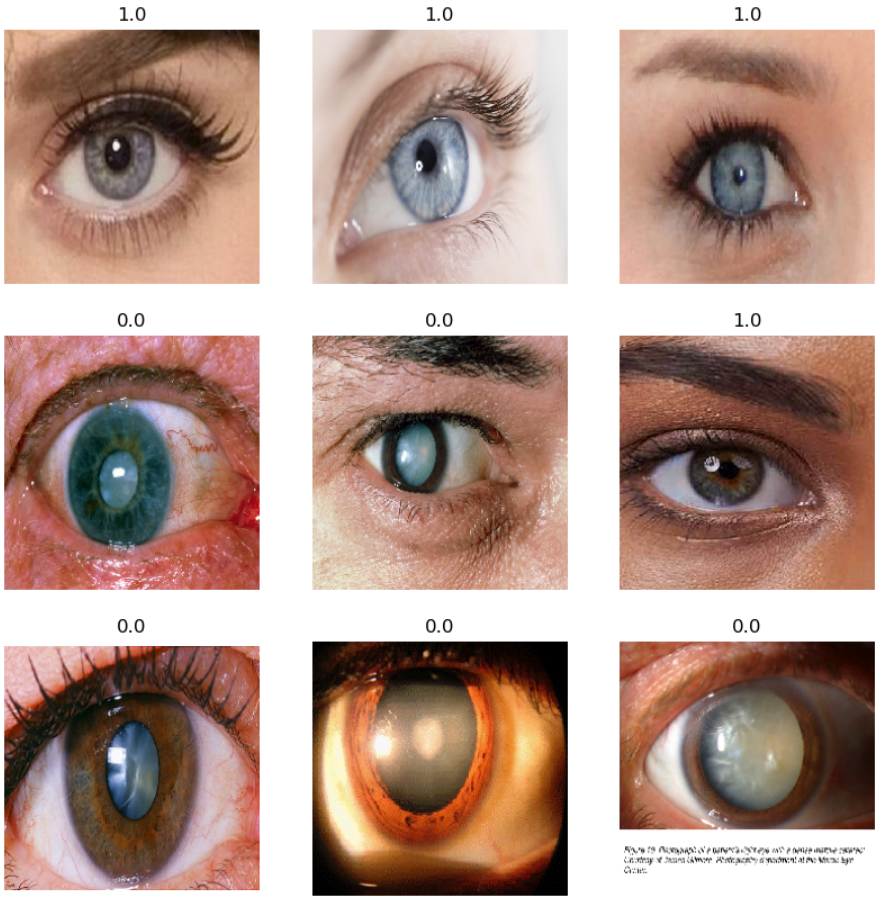
```
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try:
                files = glob.glob(path+"/"+category+"/*")
                for f in files:
                    img_array = cv2.imread(f)
                    new_array = cv2.resize(img_array, (IMG_SIZE,
IMG_SIZE))
                    data.append([np.array(img_array),
CATEGORIES.index(category)])
            except Exception as e:
                pass
    np.random.shuffle(data)
    return data
```

#For Image augumentation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator


train_data_dir = "/Users/kartiksolanki/Desktop/eye_cataract/dataset/
train"
test_data_dir = "/Users/kartiksolanki/Desktop/eye_cataract/dataset/test"


train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=20,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale = 1.0/255.)

train_generator = train_datagen.flow_from_directory(train_data_dir,
                                                    target_size=(224, 224),
                                                    batch_size=32,
                                                    class_mode='binary')


test_generator = test_datagen.flow_from_directory(test_data_dir,
                                                  target_size=(224, 224),
                                                  batch_size=32,
                                                  class_mode='binary')
```

```
Found 491 images belonging to 2 classes.
Found 118 images belonging to 2 classes.
```

```
import matplotlib.pyplot as plt

images, labels = next(test_generator)

plt.figure(figsize=(10, 10))

for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(images[i])
    plt.title(str(labels[i]) if labels[i] == 0 else str(labels[i]))
    plt.axis('off')

    plt.show()
```



#Implementing convulational Neural Network

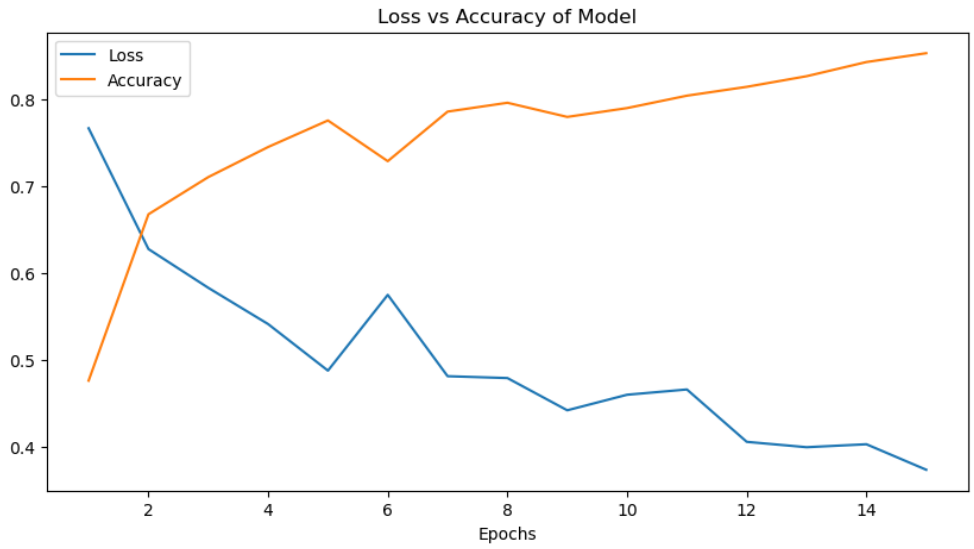
```
model = keras.Sequential([


    layers.Conv2D(32, (3, 3), input_shape=(224, 224, 3),
activation='relu'),
```

```
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(256, (3, 3), activation='relu'),#added a new
convulational layer
    layers.MaxPooling2D(2, 2),

    layers.Flatten(),
    layers.Dense(256, activation='relu'),#raised the dense layers to 256

    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')


])


model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
history = model.fit(train_generator, epochs=15,
validation_data=test_generator)
model.save('cateye-CNN.model')
```

```
16/16 [==============================] - 31s 2s/step - loss: 0.5834 - accuracy: 0.7108 - val_loss: 0.5038 - val_acc
uracy: 0.7627
Epoch 4/15
16/16 [==============================] - 31s 2s/step - loss: 0.5417 - accuracy: 0.7454 - val_loss: 0.5059 - val_acc
uracy: 0.7966
Epoch 5/15
16/16 [==============================] - 30s 2s/step - loss: 0.4881 - accuracy: 0.7760 - val_loss: 0.5267 - val_acc
uracy: 0.7797
Epoch 6/15
16/16 [==============================] - 31s 2s/step - loss: 0.5753 - accuracy: 0.7291 - val_loss: 0.5404 - val_acc
uracy: 0.7373
Epoch 7/15
16/16 [==============================] - 31s 2s/step - loss: 0.4817 - accuracy: 0.7862 - val_loss: 0.5661 - val_acc
uracy: 0.7966
Epoch 8/15
16/16 [==============================] - 29s 2s/step - loss: 0.4796 - accuracy: 0.7963 - val_loss: 0.5311 - val_acc
uracy: 0.7458
Epoch 9/15
16/16 [==============================] - 31s 2s/step - loss: 0.4425 - accuracy: 0.7800 - val_loss: 0.5341 - val_acc
uracy: 0.7966
Epoch 10/15
16/16 [==============================] - 30s 2s/step - loss: 0.4604 - accuracy: 0.7902 - val_loss: 0.3846 - val_acc
uracy: 0.8305
Epoch 11/15
16/16 [==============================] - 30s 2s/step - loss: 0.4664 - accuracy: 0.8045 - val_loss: 0.4712 - val_acc
uracy: 0.8051
Epoch 12/15
16/16 [==============================] - 31s 2s/step - loss: 0.4062 - accuracy: 0.8147 - val_loss: 0.4133 - val_acc
uracy: 0.8305
Epoch 13/15
16/16 [==============================] - 30s 2s/step - loss: 0.4000 - accuracy: 0.8269 - val_loss: 0.3812 - val_acc
uracy: 0.8559
Epoch 14/15
16/16 [==============================] - 29s 2s/step - loss: 0.4034 - accuracy: 0.8432 - val_loss: 0.4262 - val_acc
uracy: 0.7966
Epoch 15/15
16/16 [==============================] - 30s 2s/step - loss: 0.3741 - accuracy: 0.8534 - val_loss: 0.3742 - val_acc
uracy: 0.8390
```

```
import matplotlib.pyplot as plt


epochs = range(1, 16)
plt.figure(figsize=(10, 5))
plt.title("Loss vs Accuracy of Model")
plt.plot(epochs, history.history['loss'][:15], label='Loss')
```

```
plt.plot(epochs, history.history['accuracy'][:15], label='Accuracy')
plt.grid()
plt.xlabel("Epochs")
plt.grid()
plt.legend()
plt.show()
```




```
for i, (test_images, true_labels) in enumerate(test_generator):
    predictions = model.predict(test_images)
    binary_predictions = [1 if pred > 0.5 else 0 for pred in predictions]
    true_labels = [int(label) for label in true_labels]

    for true_label, binary_prediction in zip(true_labels,
binary_predictions):
        label_string = 'Cataract' if true_label == 1 else 'Normal'
        prediction_string = 'Cataract' if binary_prediction == 1 else
'Normal'
        print(f"True: {label_string}, Predicted: {prediction_string}")

    if i == len(test_generator) - 1:
        break
```

```
1/1 [==============================] - 0s 498ms/step
True: Normal, Predicted: Normal
True: Cataract, Predicted: Cataract
True: Cataract, Predicted: Cataract
True: Cataract, Predicted: Cataract
True: Normal, Predicted: Normal
True: Cataract, Predicted: Cataract
True: Cataract, Predicted: Cataract
True: Cataract, Predicted: Cataract
True: Normal, Predicted: Normal
True: Normal, Predicted: Normal
True: Cataract, Predicted: Normal
True: Normal, Predicted: Normal
True: Normal, Predicted: Normal
True: Cataract, Predicted: Cataract
True: Cataract, Predicted: Cataract
True: Cataract, Predicted: Normal
True: Normal, Predicted: Normal
True: Normal, Predicted: Normal
```

```
#Prediciting Random samples
```

```
test_image_path = 'normal.png'
img = image.load_img(test_image_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

prediction = model.predict(img_array)

binary_prediction = 1 if prediction[0] > 0.5 else 0

label_string = 'Normal' if binary_prediction == 1 else 'Cataract'
print(f"Predicted: {label_string}”)
```

```
1/1 [==============================] - 0s 24ms/step
Predicted: Normal
```

```
test_image_path = 'cataract.jpeg'
img = image.load_img(test_image_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

prediction = model.predict(img_array)

binary_prediction = 1 if prediction[0] > 0.5 else 0

label_string = 'Normal' if binary_prediction == 1 else 'Cataract'
print(f"Predicted: {label_string}”)
```

```
1/1 [==============================] - 0s 28ms/step
Predicted: Cataract
```