

SAMSUNG

Samsung Innovation Campus

| Artificial Intelligence Course

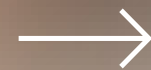
Together We Transform
Enabling People
to Achieve More

Supervised By Dr shady Nagy

Detect Retina Damage using Deep Learning

Represented By Menna ,Mostafa,Mario

Facilitator: Eng ALi Hamed



Agenda

About

**Problem
Statement**

1

Objective

Visualization

preprocessing

**Modeling and
Evaluation**

Deployment

Conclusion

Recommendation

About :

The retina is the innermost layer of the eye and it features many light-sensitive photoreceptor cells.

These cells detect light and convert it into electrical signals, which travel through the optic nerve to the brain, resulting in sight.

Retinal disorders affect the retina and typically result in visual problems.

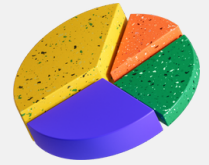


About :

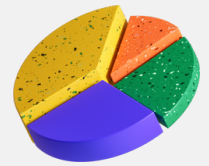
- **DRUSEN:** Eye contains yellow deposits under the retina
- **CNV :** Presence of intraretinal or subretinal fluid,
- **NORMAL:** Eye is in normal condition
- **DME:** Diabetic macular edema (DME) is a major cause of visual loss in the patients with diabetic retinopathy.



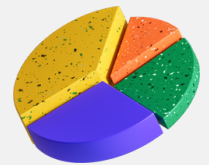
problem statment



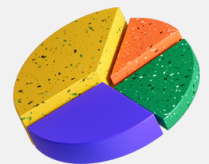
The main objective of this study is to identify what type of retinal disorder the patient have



Objective:building a system for Retina Damage using deep learning



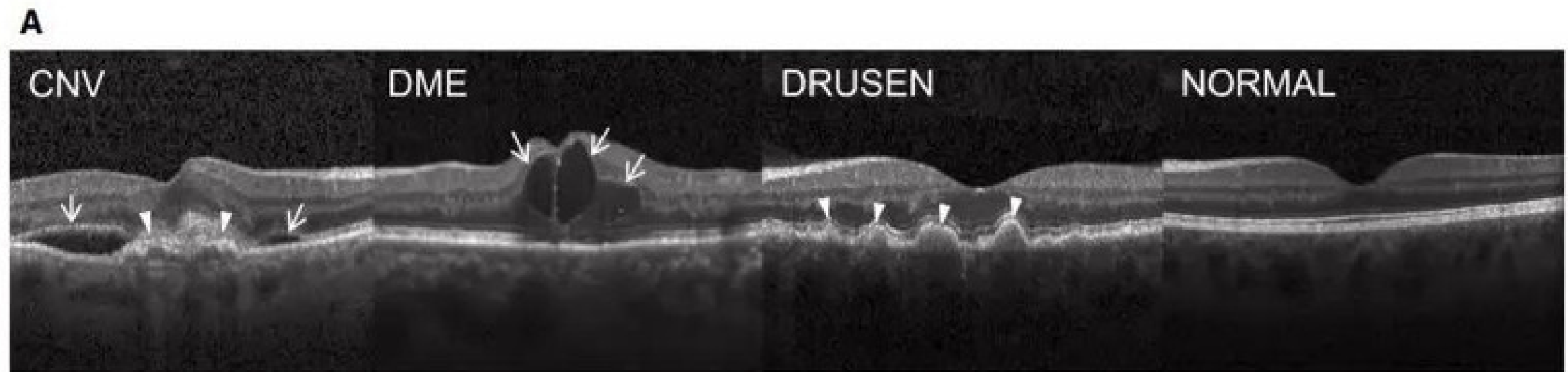
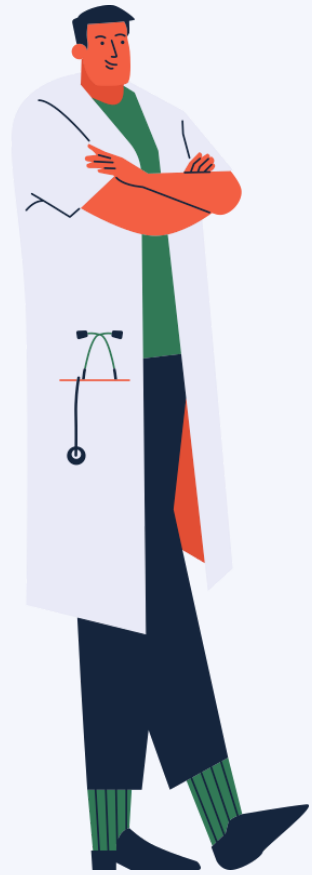
Data: contains 84.000 Retinal OCT Images



Technique: Retinal optical coherence tomography (OCT) is an imaging technique used to capture high-resolution cross sections of the retinas of living patients



Data information

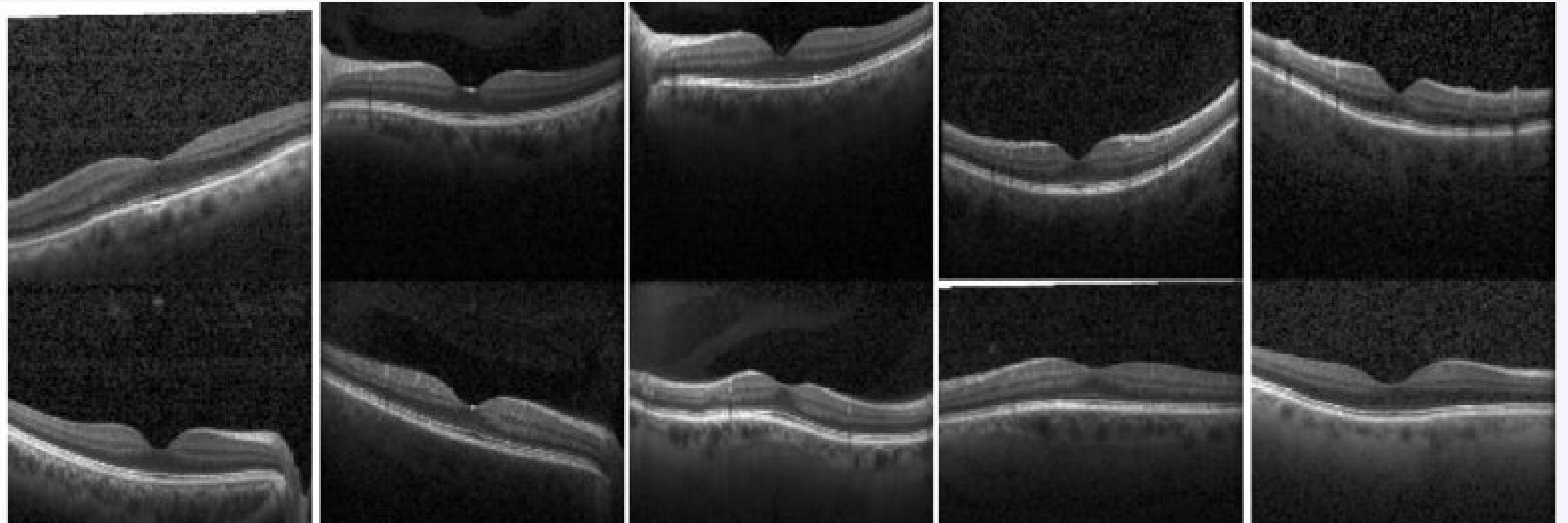


the data have 4 labels (disorders):

- choroidal neovascularization (CNV)
- Diabetic macular edema (DME)
- Multiple drusen(DRusen)
- Normal retina

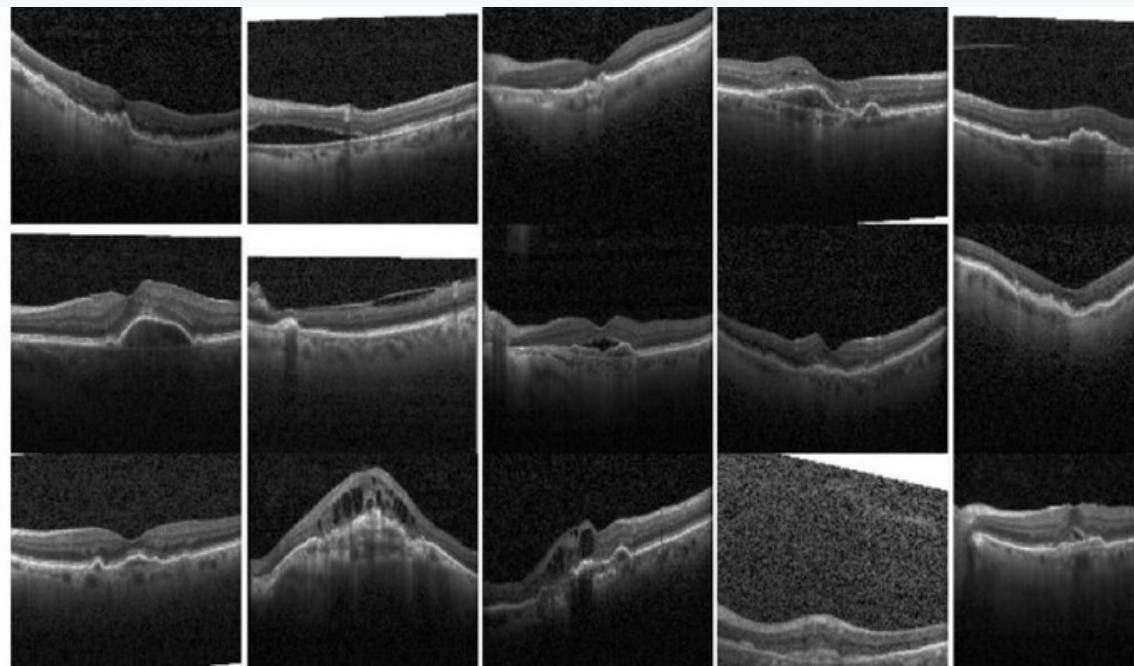
let's explore our data

Normal Patches :



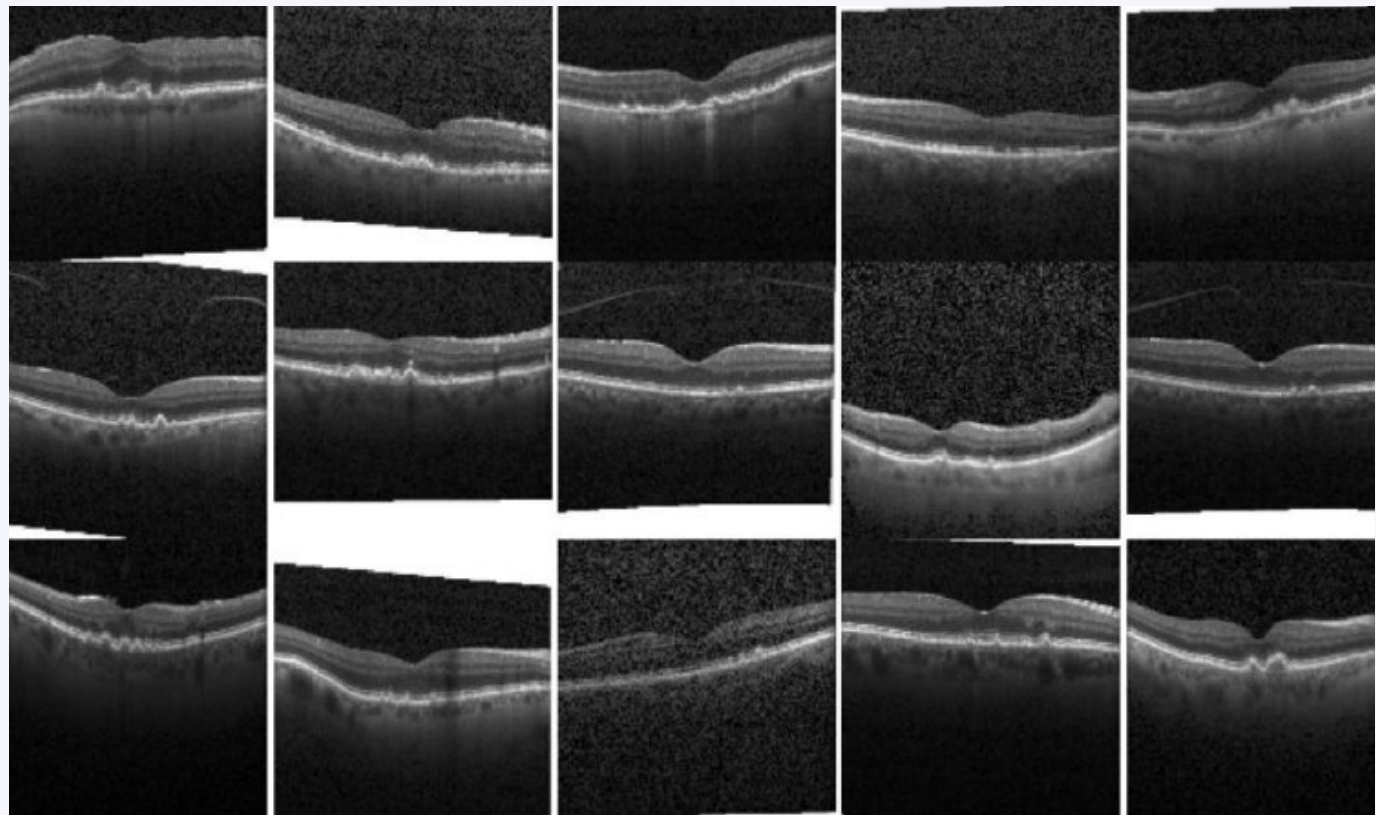
Visualization

Abnormal patches :

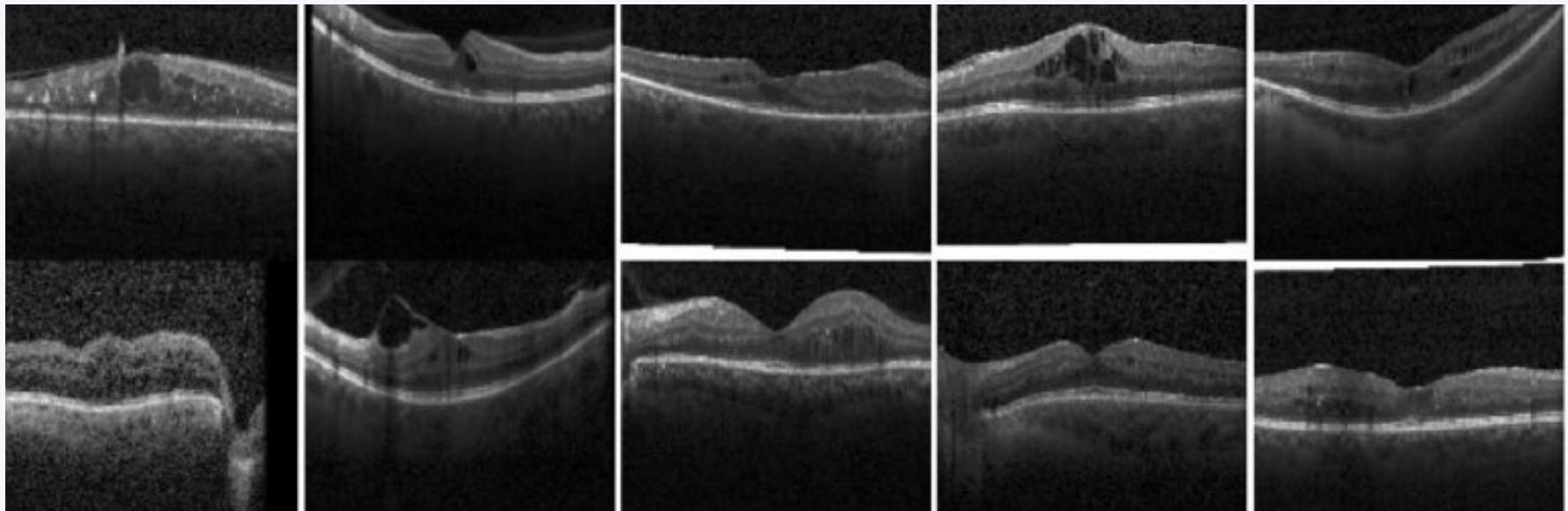


Multiple drusen(DRusen)

Diabetic macular edema (DME)



CNV



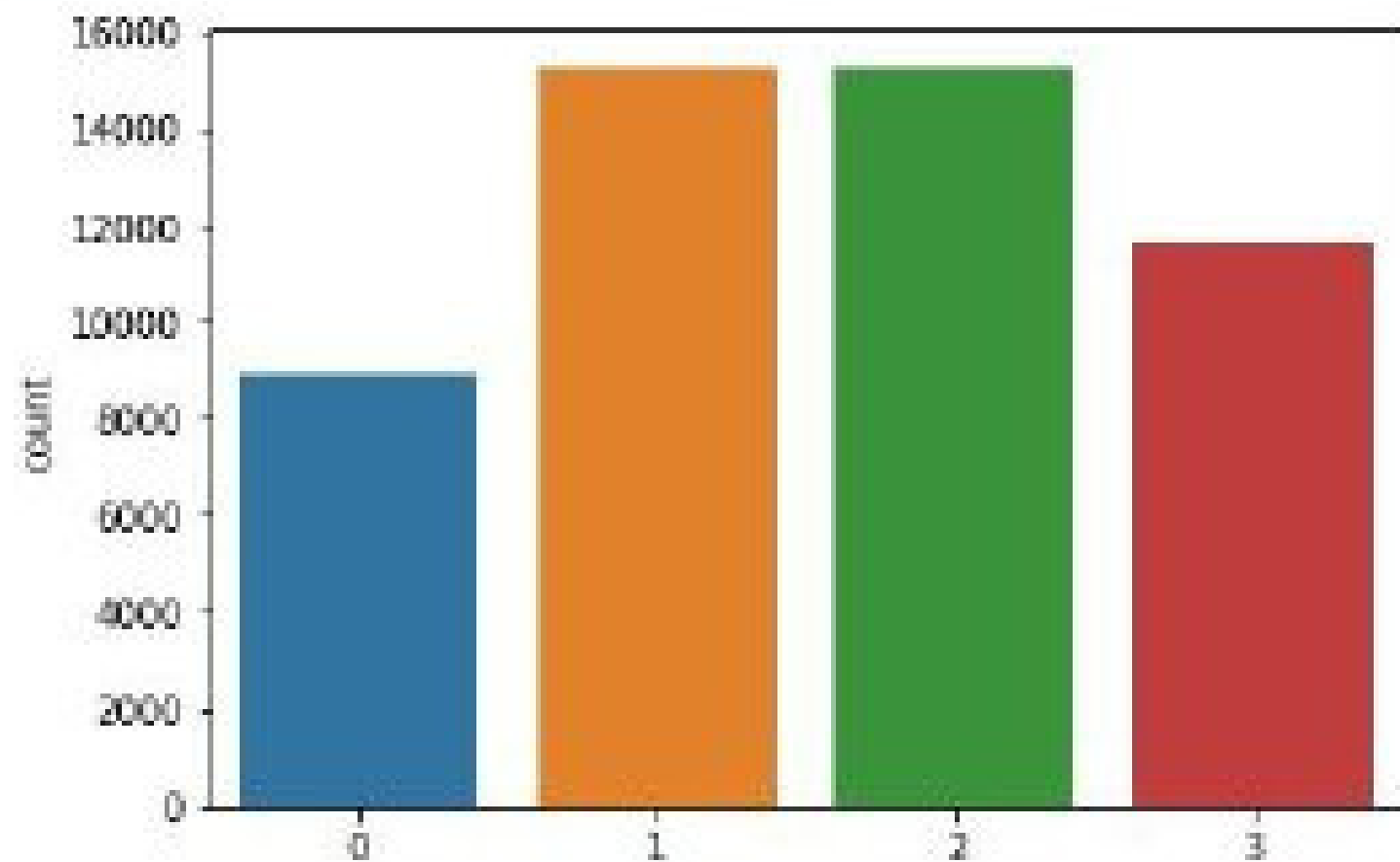
Preprocessing



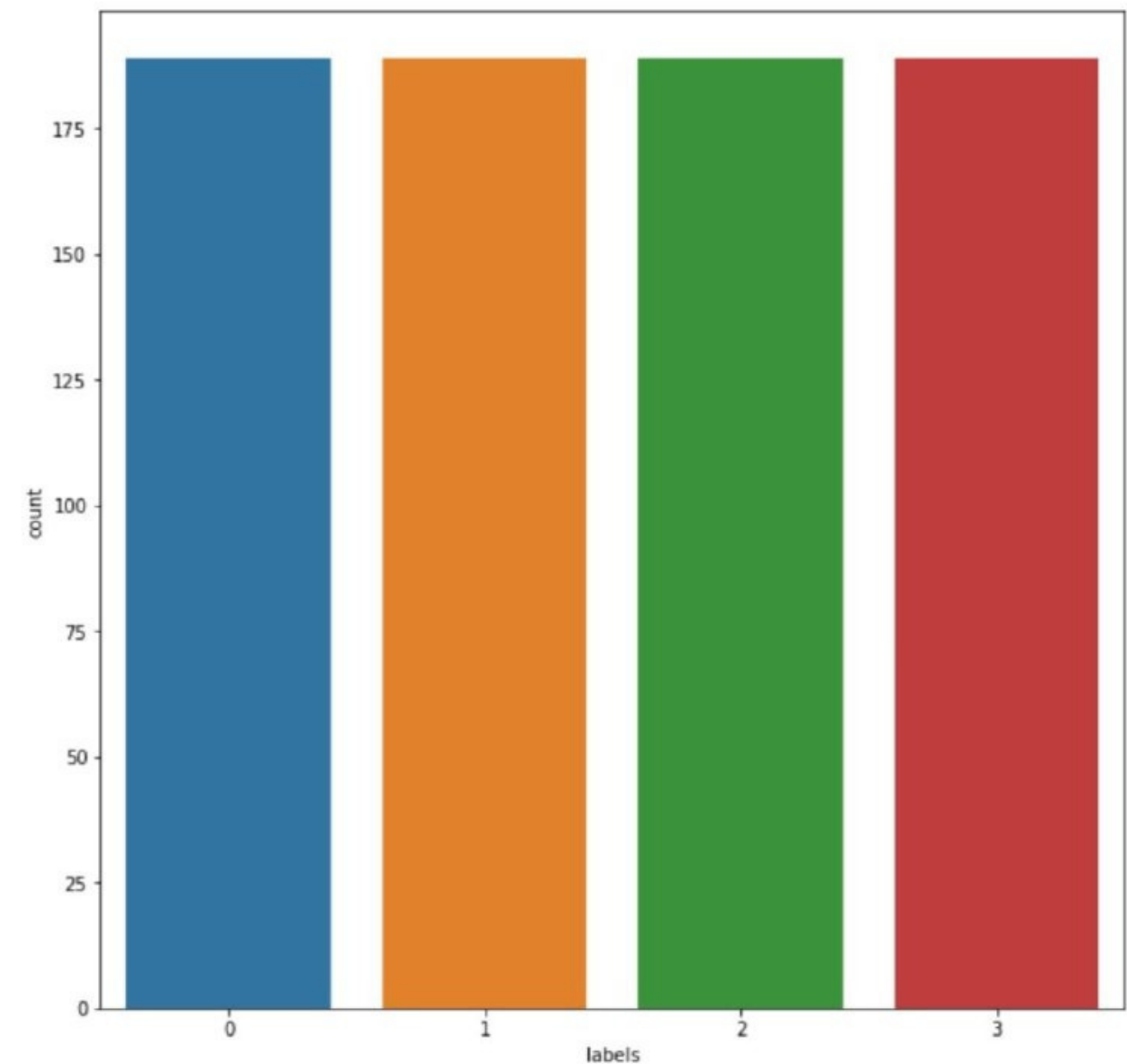
the classes were imbalanced so we made them have the same level of balance

[6]:

```
sns.countplot(labels);  
plt.show()
```



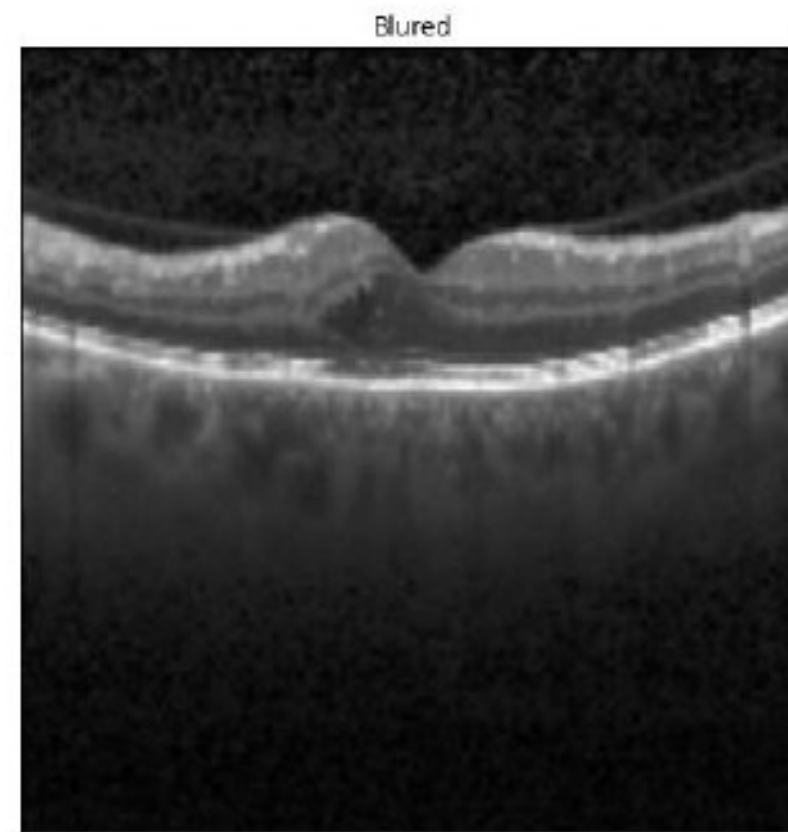
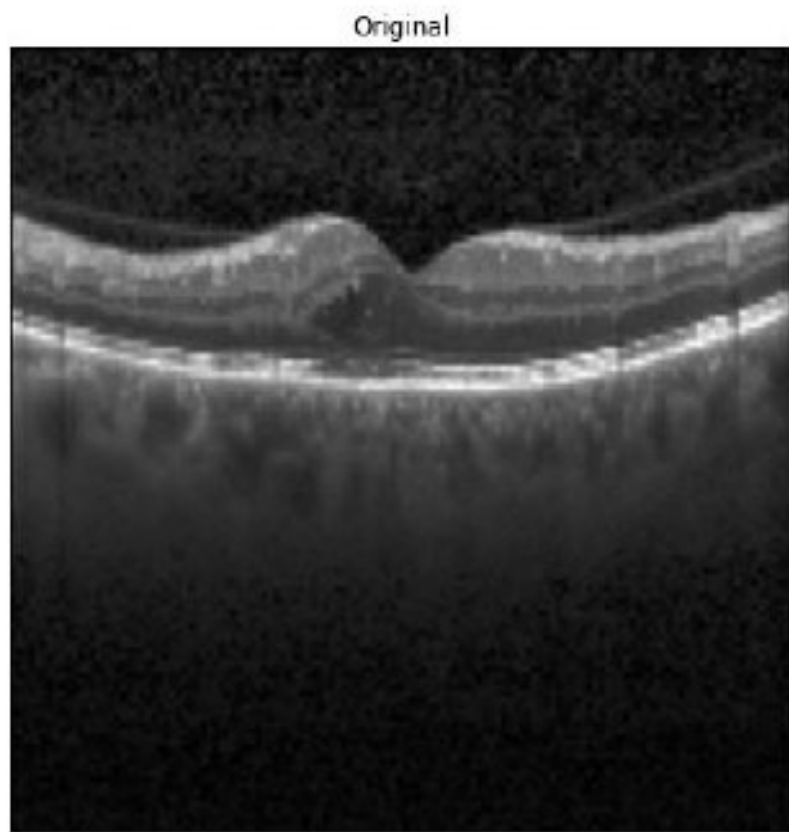
{0: 'Normal', 1: 'CNV', 2: 'DME', 3: 'DRUSEN'}



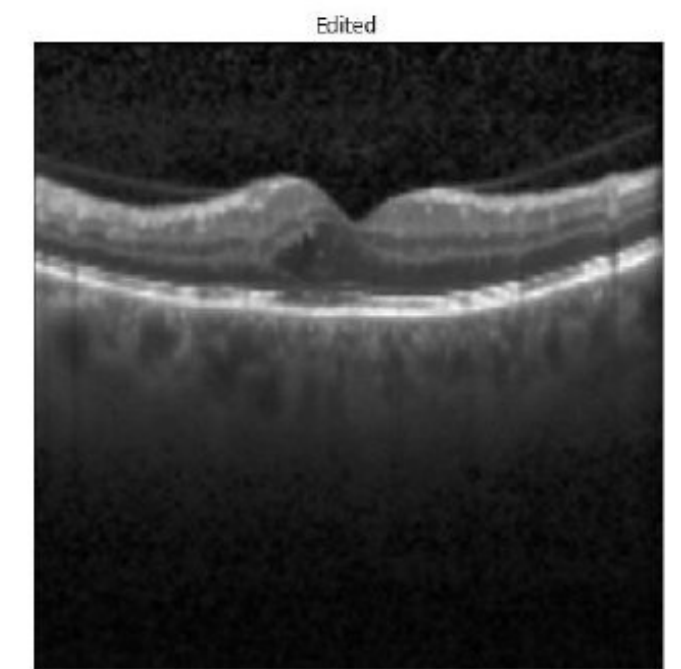
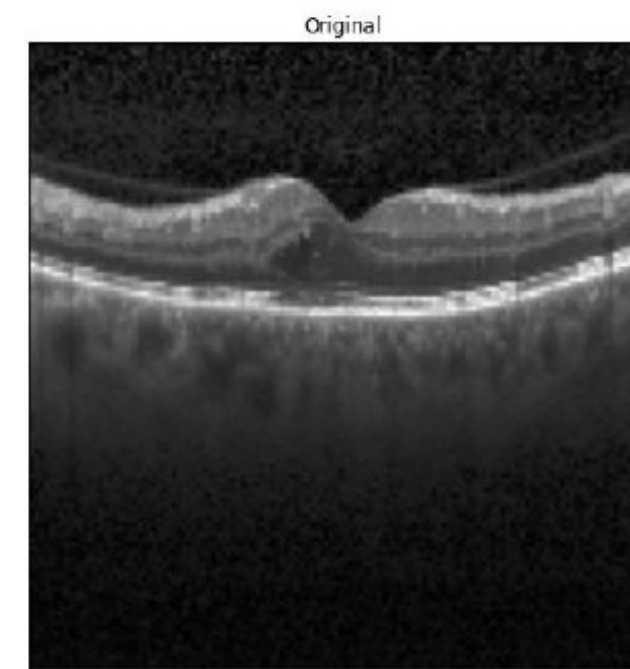
Preprocessing

- we compared between the images before and after applying the filters and as we see we lost a lot of information

```
image = no_noise[1]  
display(train_images, res_img1, 'Original', 'Blured')
```

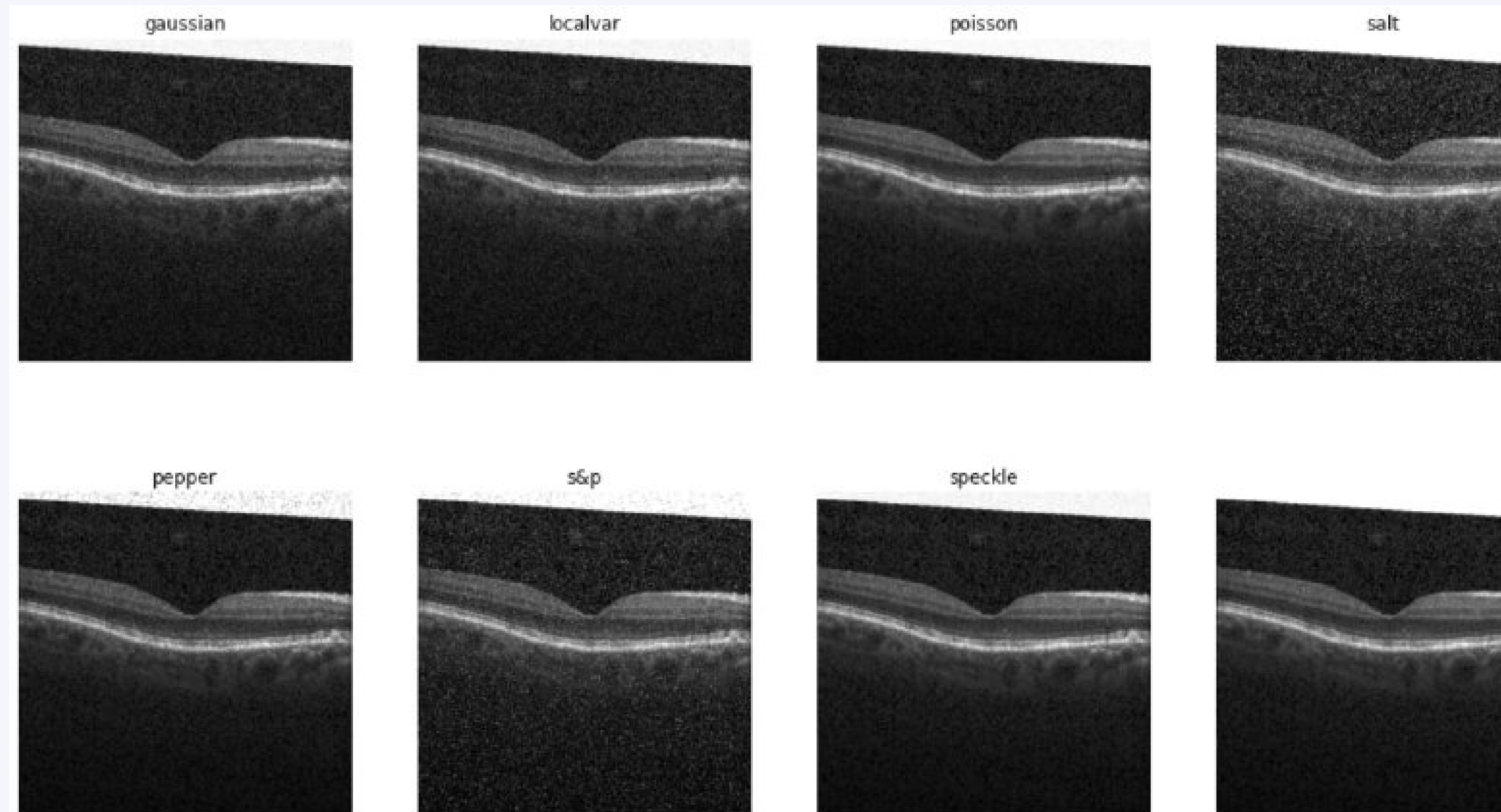


```
def display_one(train_images, title1 = "Original"):  
    plt.imshow(train_images[0]), plt.title(title1)  
    plt.xticks([], plt.yticks([]))  
    plt.show()  
# Display two images  
def display(train_images, res_img, title1 = "Original", title2 = "Edited"):  
    plt.subplot(121), plt.imshow(train_images[0], cmap=plt.cm.gray), plt.title(title1)  
    plt.xticks([], plt.yticks([]))  
    plt.subplot(122), plt.imshow(res_img[0], cmap=plt.cm.gray), plt.title(title2)  
    plt.xticks([], plt.yticks([]))  
    plt.show()  
display(train_images, res_img1, title1 = "Original", title2 = "Edited")
```



Preprocessing

we compared between the images applying many types of filters :



as we see we lost a lot of information

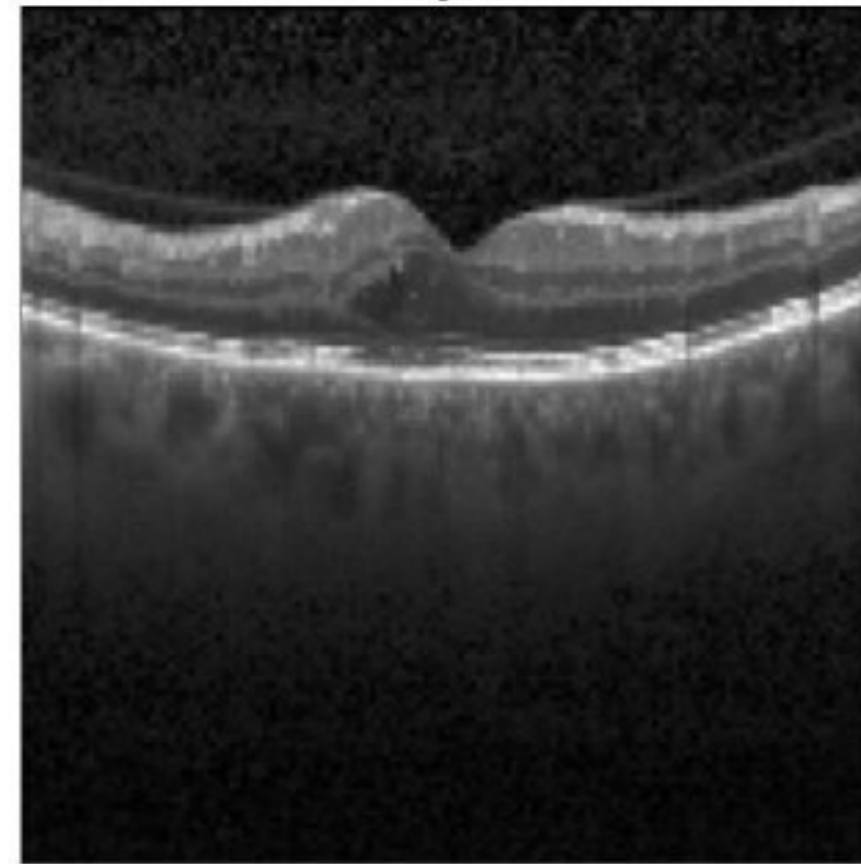
Preprocessing

- we tried to increase the brightness of each photo but we noticed that we missed part of the information so we didnt apply it in our data

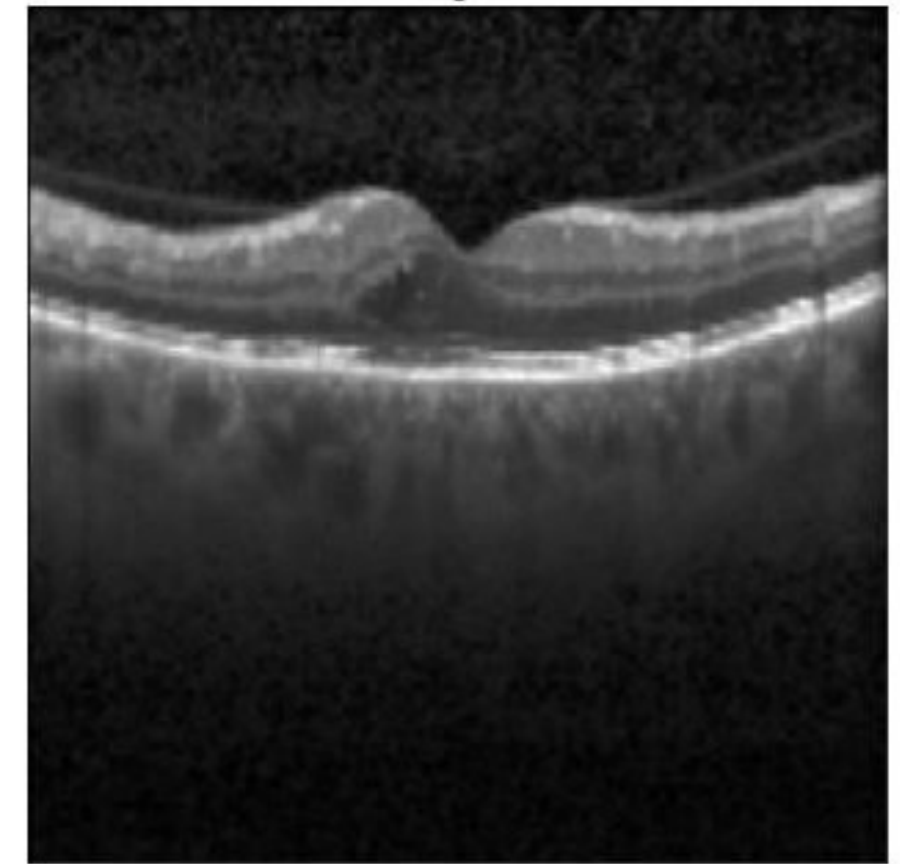
In [21]:

```
tf.image.adjust_brightness(  
    res_img1, delta=0.1)  
display(train_images, res_img1, 'Original', 'bright')
```

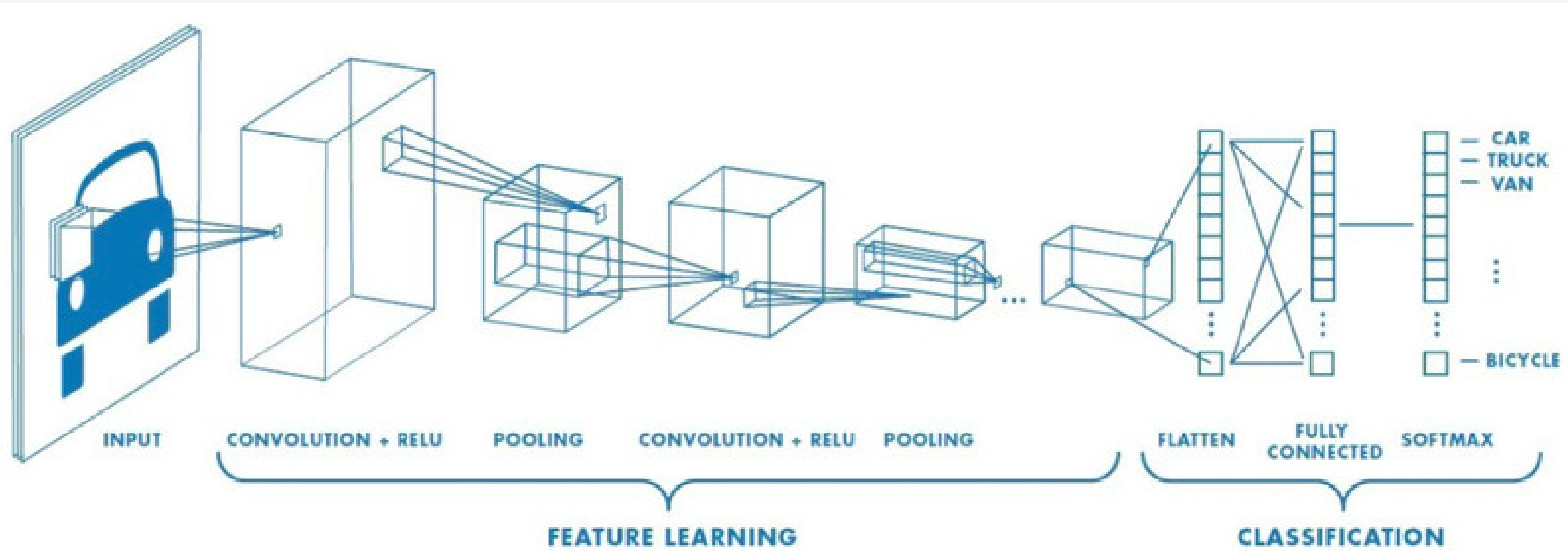
Original



bright



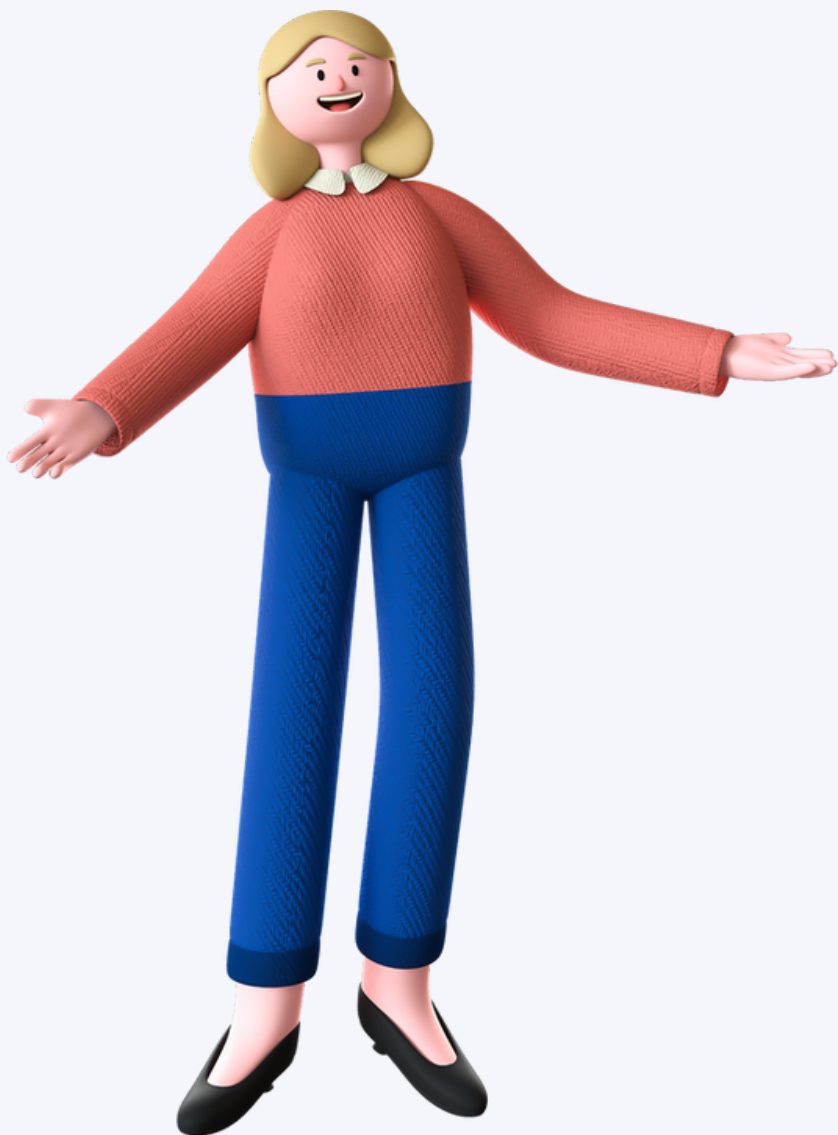
Modeling



Modeling

- **We implemented Convolutional Neural Networks (CNN)**
- **We tried more than one design and they had problems with overfitting and we solved it using dropout layer**
- **The data was divided into 80% training and 20% testing, then we passed it to the model**

Model Summary :



max_pooling2d_6	(MaxPooling2 (None, 10, 10, 60))	0
conv2d_18	(Conv2D (None, 8, 8, 30))	16230
conv2d_19	(Conv2D (None, 6, 6, 30))	8130
max_pooling2d_7	(MaxPooling2 (None, 3, 3, 30))	0
dropout_6	(Dropout (None, 3, 3, 30))	0
flatten_1	(Flatten (None, 270))	0
dense_1	(Dense (None, 5000))	1355000
dropout_7	(Dropout (None, 5000))	0
dense_2	(Dense (None, 4))	20004

Total params: 1,651,424		
Trainable params: 1,651,424		
Non-trainable params: 0		

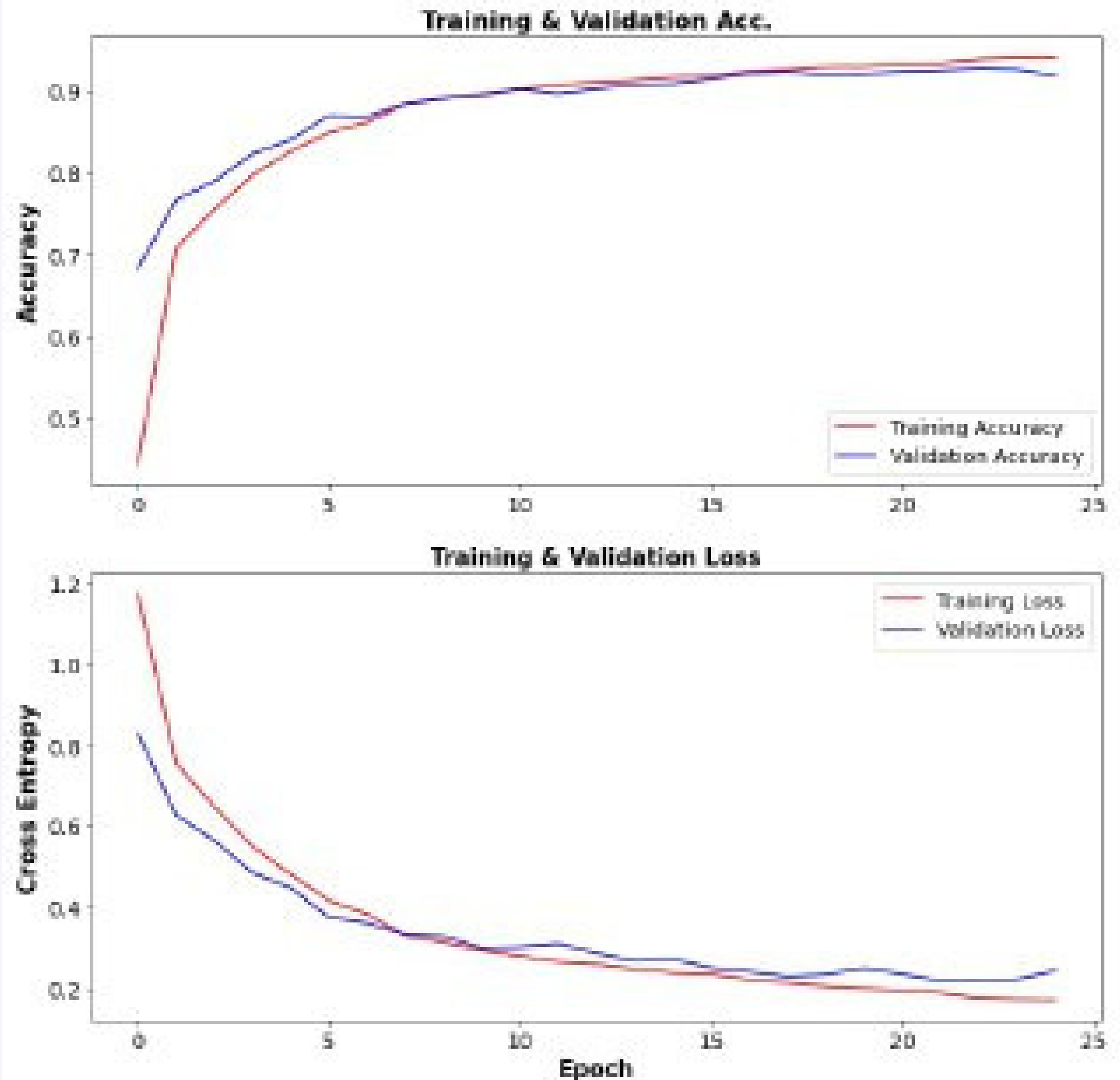
Modeling

Applying model on preprocessed images, give results:

```
Epoch 21/25
112/112 [=====] - 81s 720ms/step - loss: 0.1727 - accuracy: 0.9392 - val_loss: 0.2174 - val_a
ccuracy: 0.9328
Epoch 22/25
112/112 [=====] - 81s 720ms/step - loss: 0.1739 - accuracy: 0.9404 - val_loss: 0.2297 - val_a
ccuracy: 0.9230
Epoch 23/25
112/112 [=====] - 81s 719ms/step - loss: 0.1708 - accuracy: 0.9407 - val_loss: 0.2353 - val_a
ccuracy: 0.9262
Epoch 24/25
112/112 [=====] - 80s 719ms/step - loss: 0.1589 - accuracy: 0.9446 - val_loss: 0.2215 - val_a
ccuracy: 0.9284
Epoch 25/25
112/112 [=====] - 80s 718ms/step - loss: 0.1495 - accuracy: 0.9486 - val_loss: 0.2076 - val_a
ccuracy: 0.9334
```

Evaluation

**the model was good and it
didn't have fluctuations**



we tested our model on real photos using this function :

Test cases

```
image_path1 = '../input/kernasy2018/OCT2017_2/train/DME/DME-1872815-1.jpeg' #DME
image_path2 = '../input/kernasy2018/OCT2017_2/train/DME/CNV-1816842-188.jpeg' #DRUSEN

def predict(img):
    input_img = cv2.imread(img)
    input_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
    input_img_resize = cv2.resize(input_img, (128, 128))
    input_img_resize = np.expand_dims(input_img_resize, axis=2)
    input_img_resize = np.expand_dims(input_img_resize, axis=0)
    model_prediction = model.predict(input_img_resize)
    model_prediction = model_prediction.astype(int)
    return model_prediction

print(predict(image_path1))
print(predict(image_path2))

# The order of labels
# 'DRUSEN', 'CNV', 'NORMAL', 'DME'
```

```
[[0 0 0 1]]
[[1 0 0 0]]
```

+ Code

+ Markdown

True prediction for 2 random test cases

Model Outputs and Outcomes

Model Evaluation :

```
248/248 [=====] - 9s 31ms/step - loss: 0.2181 - accuracy: 0.9317  
Test Loss: 0.21813897788524628  
Test accuracy: 0.9317263960838318
```

After several attempts, we made a good model design :

- **Our model have a good acc = 93%**
- **we dont have overfitting**
- **We're ready to save our model and create APIs**



Save Model

How was the model saved :



```
In [29]: model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")
```

Saved model to disk

Deployment



Tools used in Deployment :

- **Flask Framework to build Api**
- **Html, Css to build Gui**

Deployment

Please upload your retina x-ray image

Choose File No file chosen

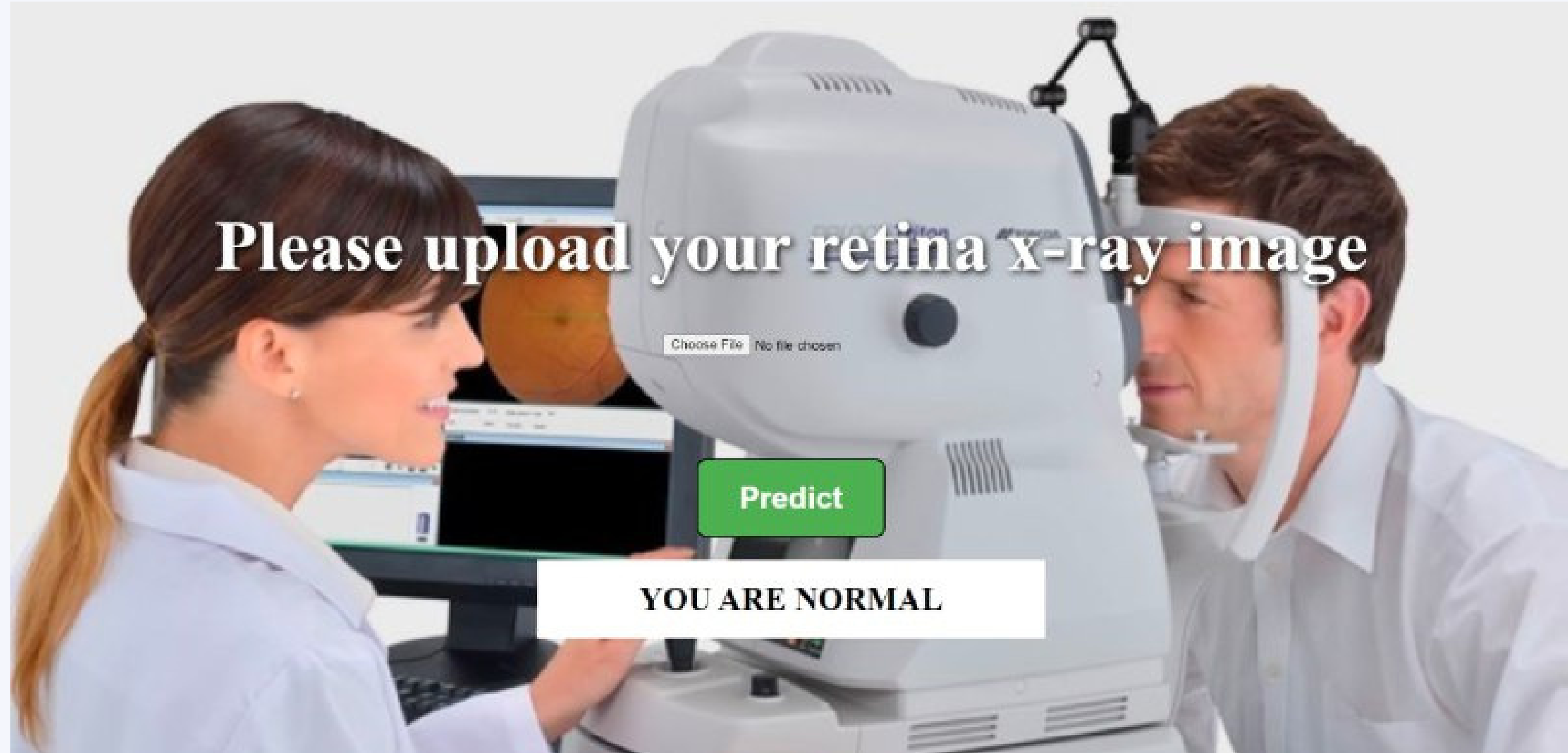
Predict

Deployment



- we upload oct image to predict the type of damage :

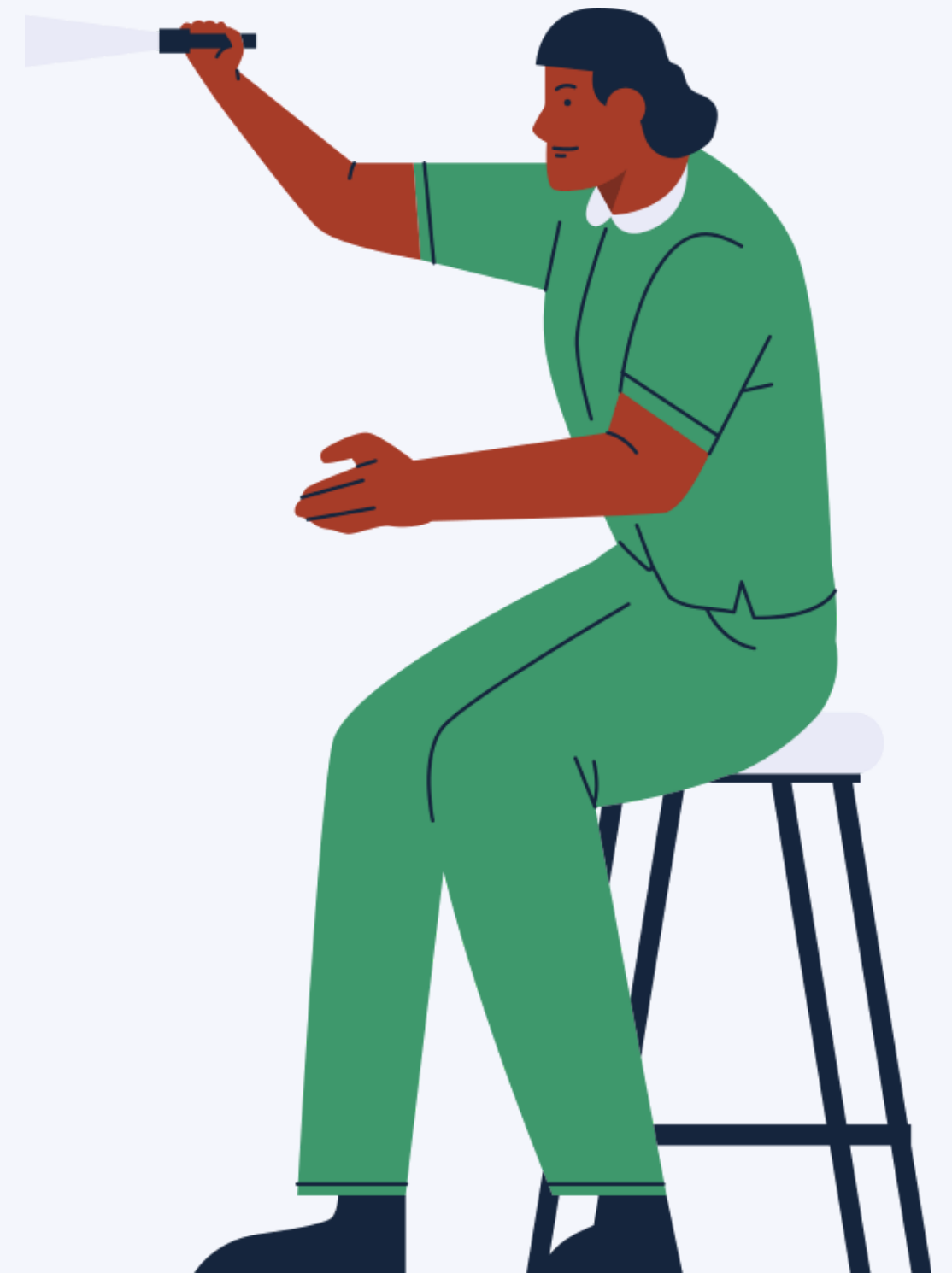
Deployment



- **the result of the oct image was normal so the patient is good**

Future work

- we tried to use SRGANS but the model needed more memory so in the future we will solve this problem
- we will continue our work with the application but with the description of each damage and what to do to heal from the damage and which behaviours he must protect himself from during the damage



GET CONTACT



EMAIL ADDRESS **mostafamenna304@gmail.com**



PHONE NUMBER **01063557277**



EMAIL ADDRESS **mostafa.ahraf154@gmail.com**



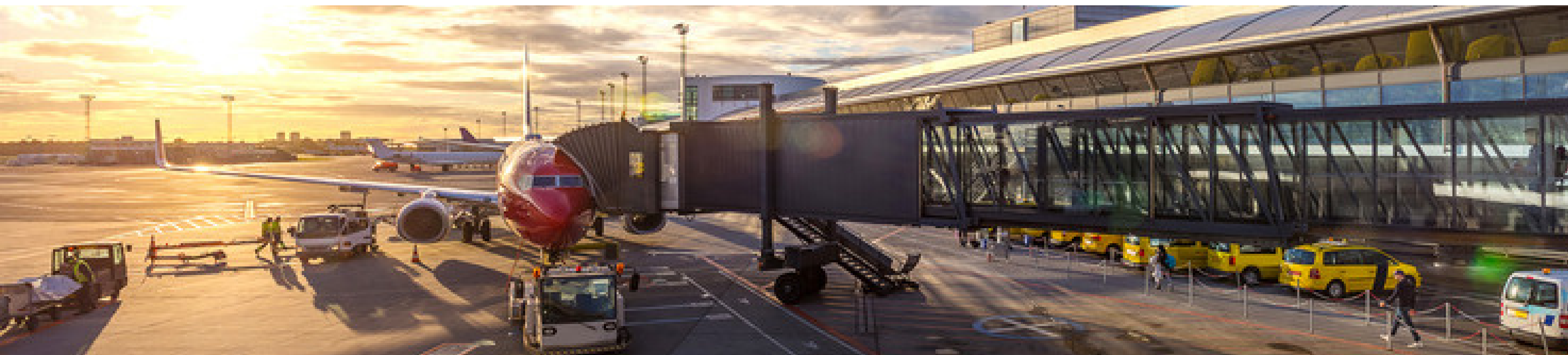
PHONE NUMBER **01283033359**



LINKEDIN **[HTTPS://WWW.LINKEDIN.COM/IN/MENNA-MOSTAFA-1AB0311B7](https://www.linkedin.com/in/menna-mostafa-1ab0311b7)**



LINKEDIN **<https://www.linkedin.com/in/mostafa-ashraf-17913a211/>**



THANK YOU!



SAMSUNG

Together for Tomorrow!
Enabling People

Education for Future Generations