



Diabetic Retinopathy

Team

Akshat Jain

Krishna Kumar Dixit

Eshaan



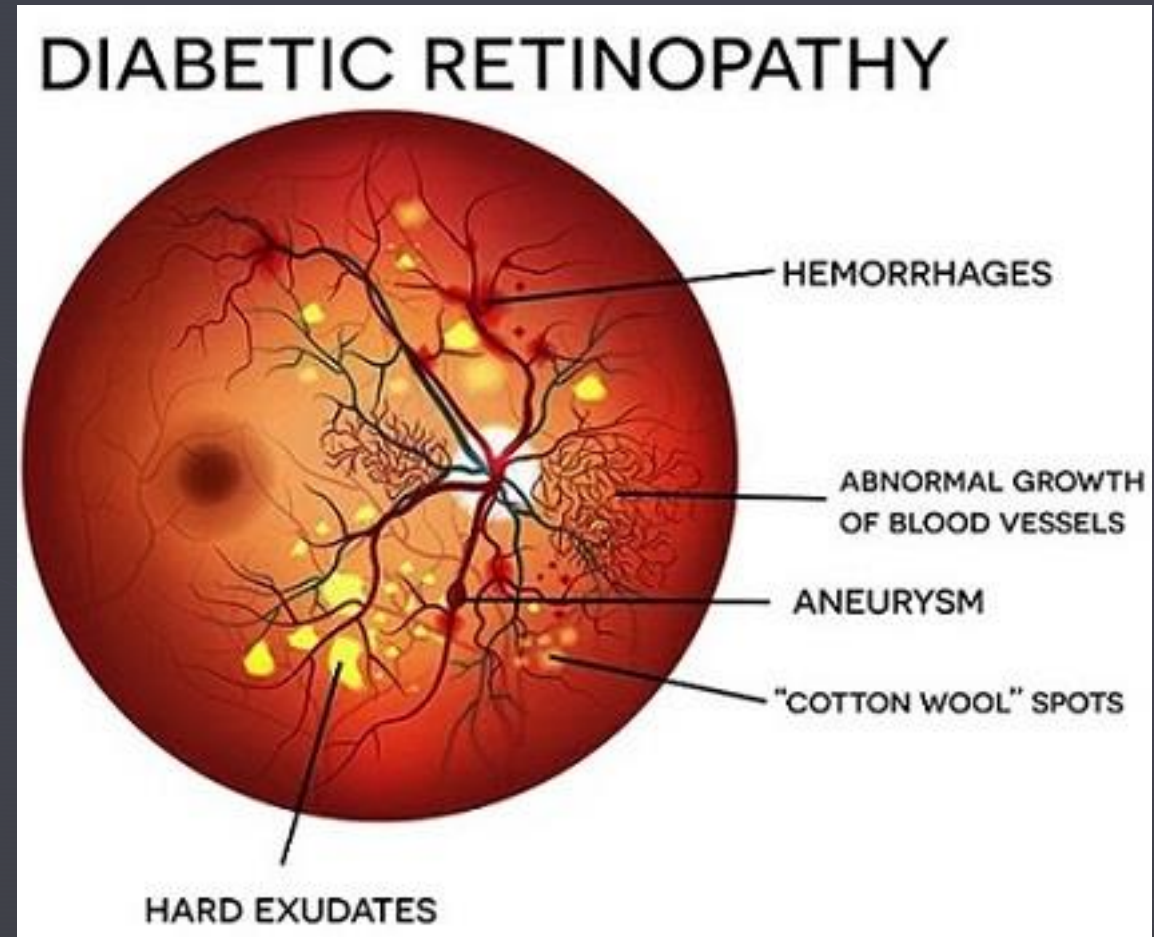
And explanation of problem statement

Introduction



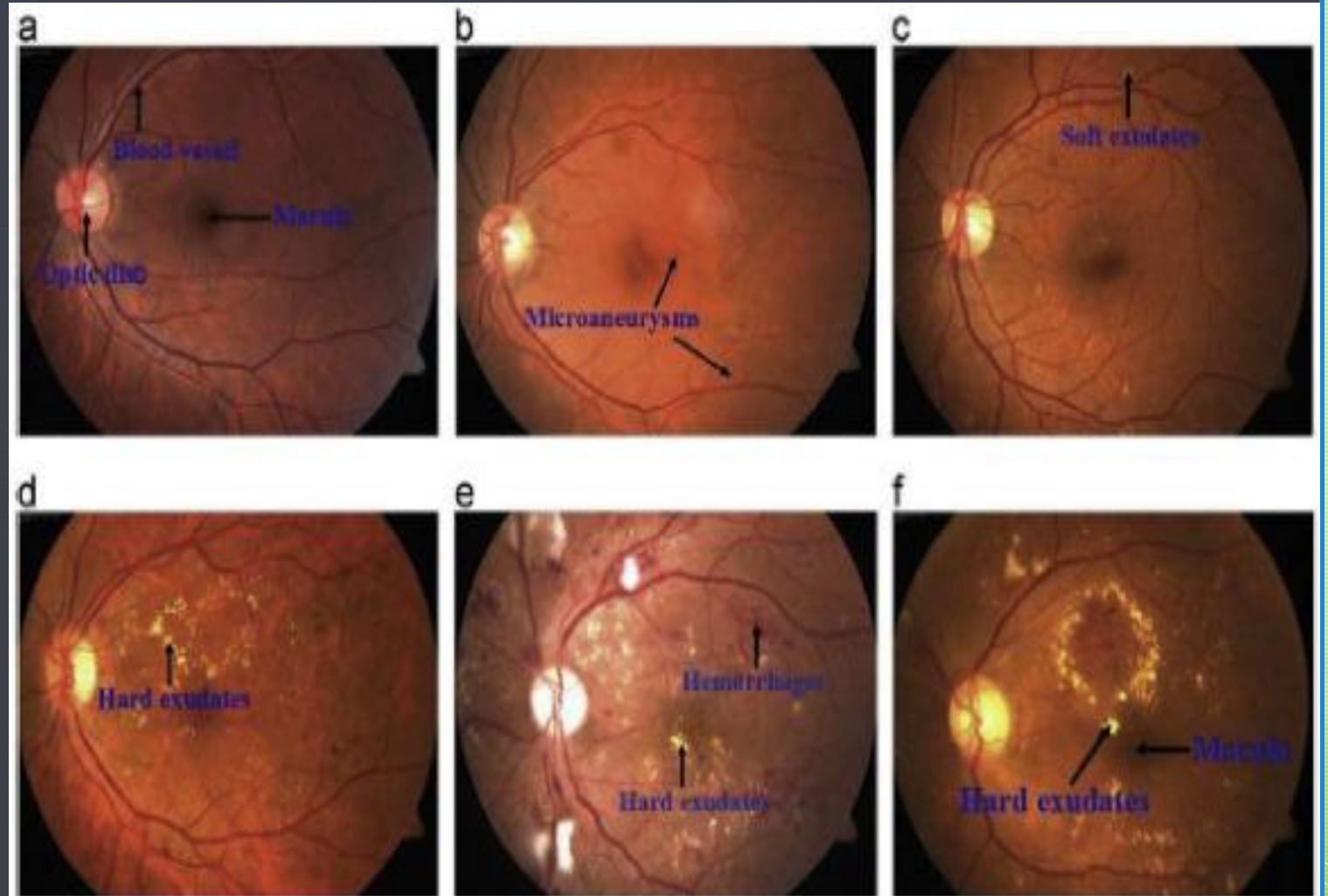
Diabetic Retinopathy

- Diabetic retinopathy is the leading cause of blindness in the working-age population of the developed world. It is estimated to affect over 93 million people.
- Diabetic Retinopathy (DR) is an eye disease associated with long-standing diabetes in which retina is damaged due to *Diabetes Mellitus*.



Features in retinal images

1. **Microaneurysms** – red dots like structures.
2. **Hemorrhages** – blood clots as flame and dot-blot like structures.
3. **Exudates** – leakage of lipoproteins and fluids due to rupture of blood vessels due to high blood pressure resulting in yellow waxy flakes.



How to classify?

Identifying features mentioned before, in the images, we classify based on severity index,

- a) 0 – no DR
- b) 1 – mild (non-proliferative)
- c) 2 – moderate (non-proliferative)
- d) 3 – severe (non-proliferative)
- e) 4 – proliferative DR



Normal eye, severity 0



Highly affected eye, severity 4

nature of the images, the
pre-processing and the noise

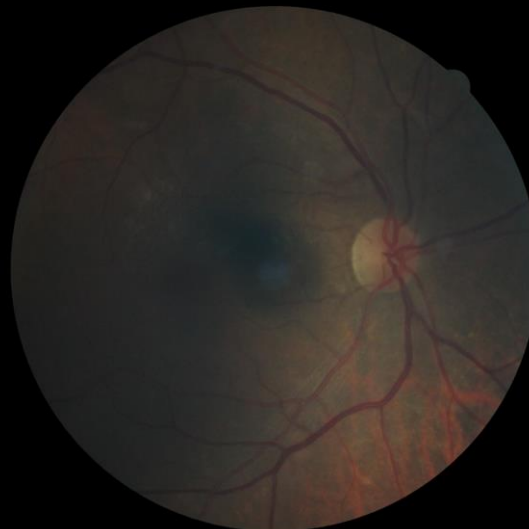
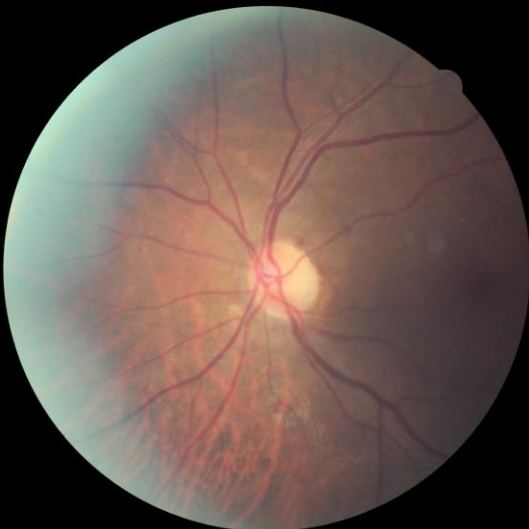
The images

About the images

- The images in the dataset come from different models and types of cameras, which can affect the visual appearance of left vs. right. Some images are shown as one would see the retina anatomically (macula on the left, optic nerve on the right for the right eye). Others are shown as one would see through a microscope condensing lens (i.e. inverted, as one sees in a typical live eye exam). There are generally two ways to tell if an image is inverted:
 - It is inverted if the macula (the small dark central area) is slightly higher than the midline through the optic nerve. If the macula is lower than the midline of the optic nerve, it's not inverted.
 - If there is a notch on the side of the image (square, triangle, or circle) then it's not inverted. If there is no notch, it's inverted.











▸ Problems with individual images

- Image of left and right eyes (lateral inversion of “right” images)
- Underexposed and Overexposed
- Circular image in a rectangular frame (crop the image)

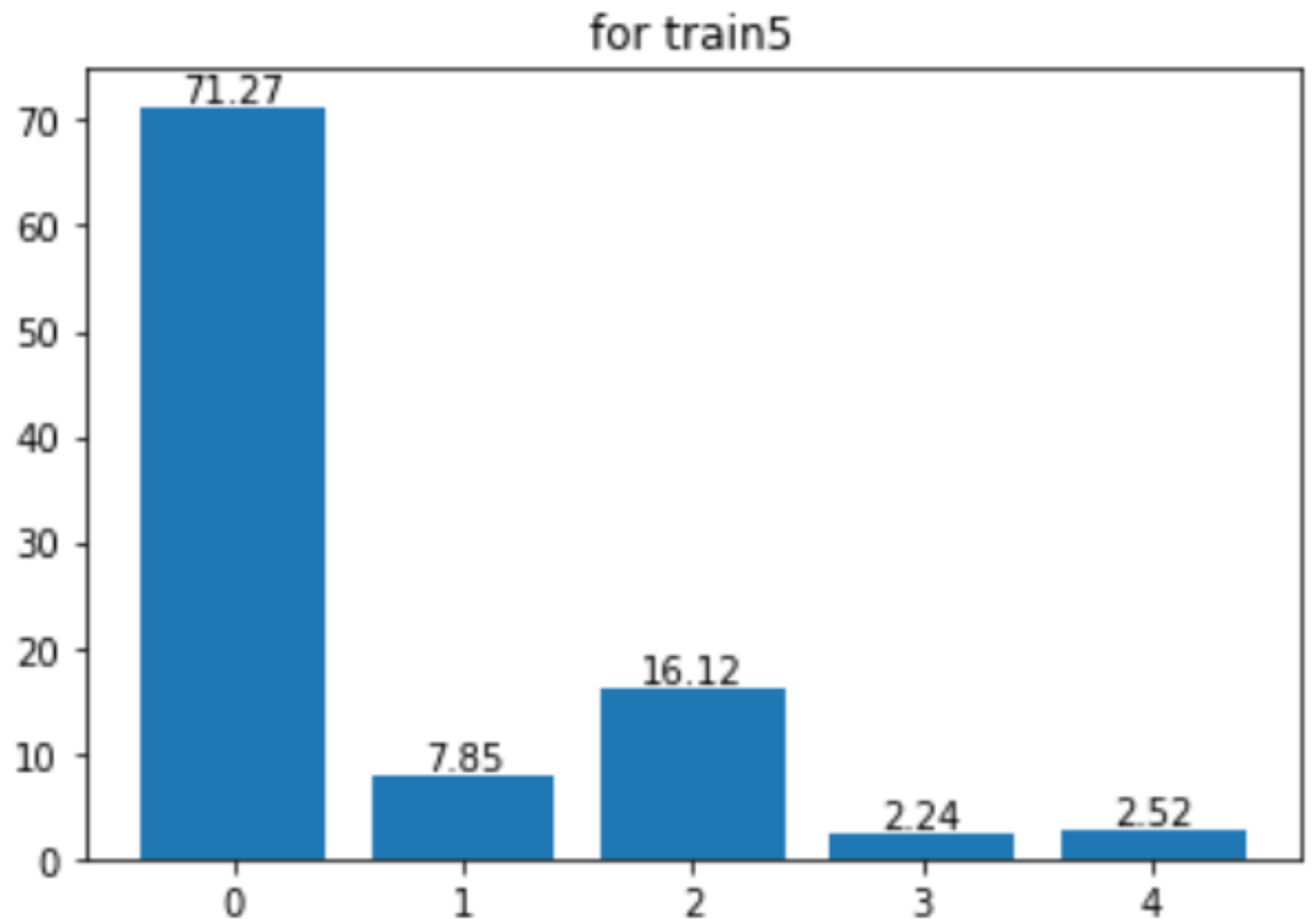


▀ Problems with image dataset

- Big data – Data is very big, so we divide it into 4 parts (with equal Distribution of severity index between parts) and train the model on each of them individually.
- Unbalanced dataset – There are many ways to handle it – undersampling, oversampling, ensemble method, fixing class weights, and a recent method of “Categorical focal loss”. We have used the Class weight method.

Name	Date modified	Type	Size
 train1	25-11-2022 22:35	File folder	
 train2	25-11-2022 22:36	File folder	
 train3	25-11-2022 22:37	File folder	
 train4	25-11-2022 22:39	File folder	
 train5	25-11-2022 22:39	File folder	
 labels1	24-11-2022 19:56	Microsoft Excel Com...	119 KB
 labels2	24-11-2022 20:20	Microsoft Excel Com...	118 KB
 labels3	24-11-2022 21:00	Microsoft Excel Com...	120 KB
 labels4	24-11-2022 21:43	Microsoft Excel Com...	115 KB
 labels5	24-11-2022 21:52	Microsoft Excel Com...	19 KB

```
: print ( test_images.shape )  
print ( test_labels.shape )  
  
(1427, 128, 128, 3)  
(1427, 5)
```



Total number of images, 1427

▀ Image Pre- Processing

- Cropping image to a square.
- Changing resolution (64X64 or 128X128)
- Overexposure and underexposure
- Contrast, etc. (Pre-made model)

Cropping and resizing (changing resolution) were done by cv2. After that, predefined pre-processing layers specific to the base model were used. No extra function was used for dealing with overexposure and underexposure

Also, a layer is made for introducing some random rotation in the images when passing through the model. (for increasing the generality of the model)

```
# for cropping the image,
def image_crop ( a ):
    # to find x1 , x2 , y1 , y2
    # to find x1, x2, y1, y2
    h = a.shape [0]
    v = a.shape [1]

    vertical = a [ :,int (v/2),:]
    for i in range ( h ):
        if (vertical [i] != [0,0,0]).any():
            y1 = i
            break

    for i in range ( h-1 , -1 , -1 ):
        if (vertical [i] != [0,0,0]).any():
            y2 = i
            break

    # to get the coordinates of x1 and x2,
    d = int ((y2 - y1) / 2)
    x1 , x2 = int (v/2 - d) , int ( v/2 + d )
    |
    a_crop = a [y1:y2 , x1:x2 , :]
    return a_crop
```

```
for j in range ( 5 ):
    name = 'train' + str (j)
    #-----
    # for images,

    resolution = 128

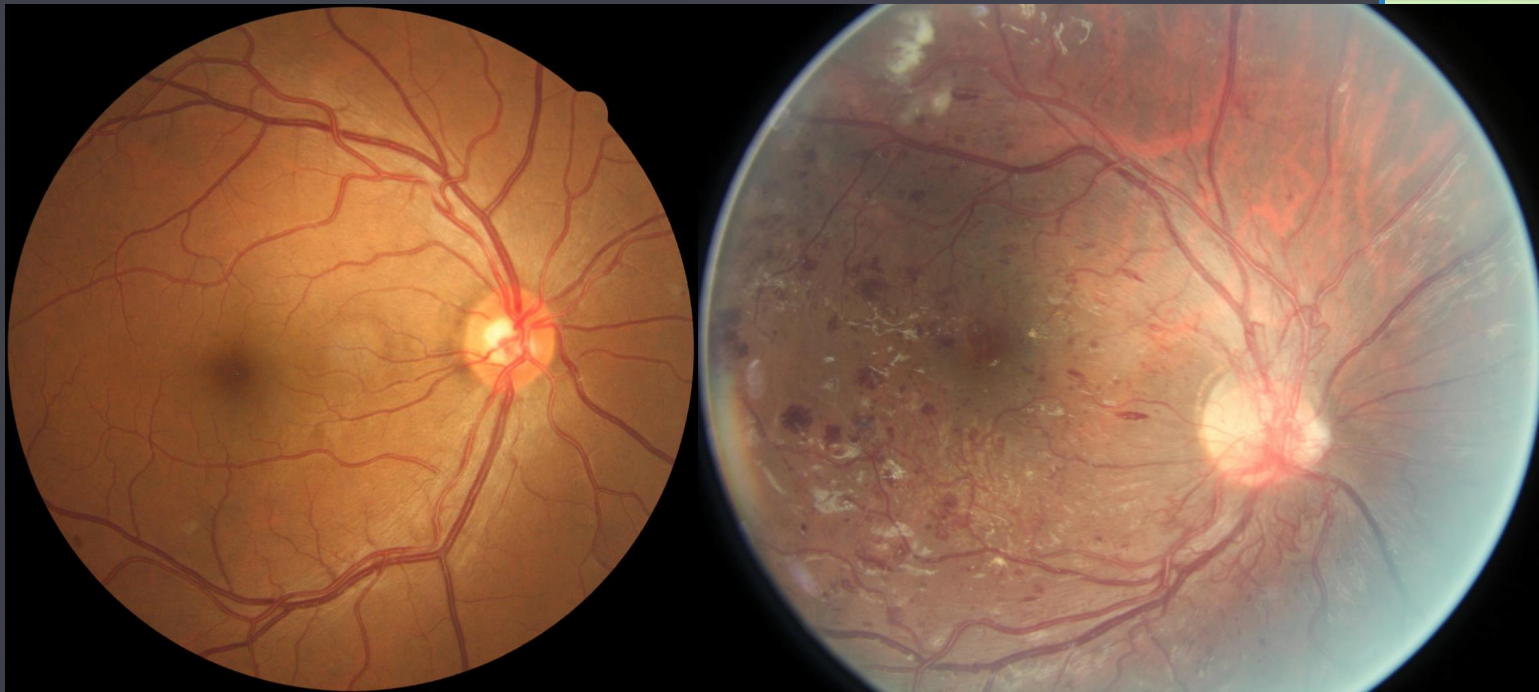
    # for reading the images, and processing them, and saving them,
    #images = np.array ( [ cv2.imread ( 'D:\\courses\\ic 272 data science 3
    #for image in os.listdir( main_path ):
    for image in os.listdir( main_path + name ):
        image_array = cv2.imread( main_path + image )
        if image [ -7 ] == 'h': # that is, if the image is 'right'
            image_array = cv2.flip( image_array , 1 )
        image_array = image_crop (image_array)
        image_array = cv2.resize ( image_array , (resolution,resolution) )
        # now, the image has been processed. Now, we just save this image.
        #images = np.append ( images , image_array , axis = 0 )
        cv2.imwrite ( save_path + name + '\\\\' + image , image_array )
        #cv2.imwrite ( save_path + image , image_array )
```

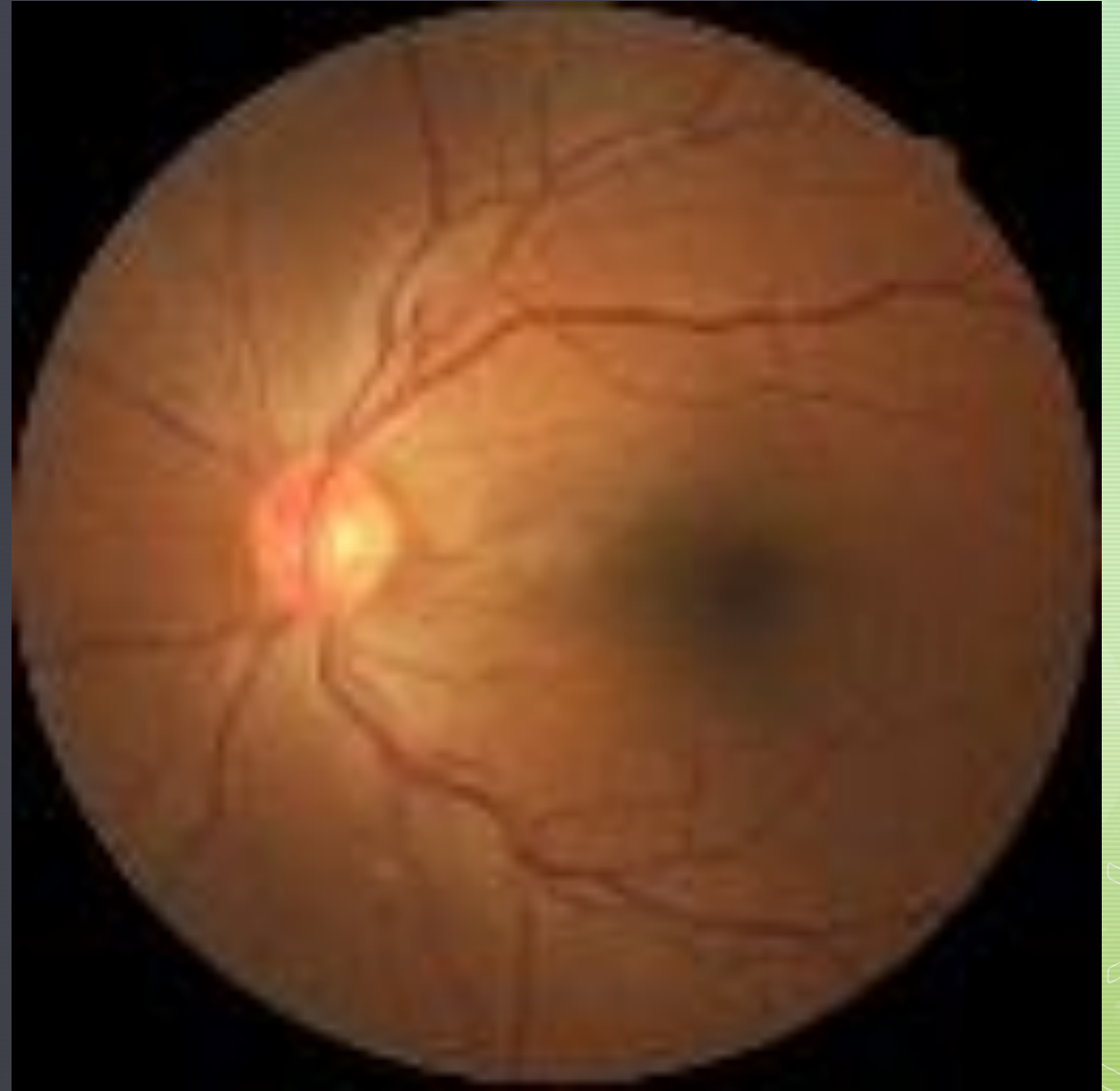
Should we use Grey scale?

There are advantages to both rgb and greyscale pictures.

- Features like hard exudates (yellow in colour) can be better identified from the rest of the features, if we use colour images.
- However, we can take a better resolution of images if we use greyscale images.

We use colour images for this model. The resolution is kept 64X64 or 128X128.







The layers involved

How to make the model?

What are the approaches?

- By designing convolutional layers, and various hyperparameters ourselves
 - The data is small, we cannot define a very complicated model, as too many weights will result in overfitting. Also, the feature extraction would not be good enough, because of lack of complexity in the model.
- By using pretrained models and trying for fine tuning
 - Can be used for smaller data. Also, contains good complexity with pretrained weights in feature extraction layers (which is helpful for feature extraction).

A generalized code

- We try to make the code write-up as general as possible. Implying, given
 - Name of the model
 - Path of images
 - Resolution of images
 - Pre-processing code

We could make another notebook for these specifications

Transfer learning, what to do?

1. Freezing base model completely.
2. Freezing most of the initial layers of the base model. (fine-tuning)

“2” is not done if the data is very small. However, if the data is large enough,

- We use “1” if the pre-trained model was trained on extracting mostly similar features.
- We use “2” if the pre-trained model extracts extremely generalized features, which are not useful for our particular case.

It may be noted that, for datasets large enough, it is preferable to do “1” first and “2” after that, for the same model

▀ Choice of base model

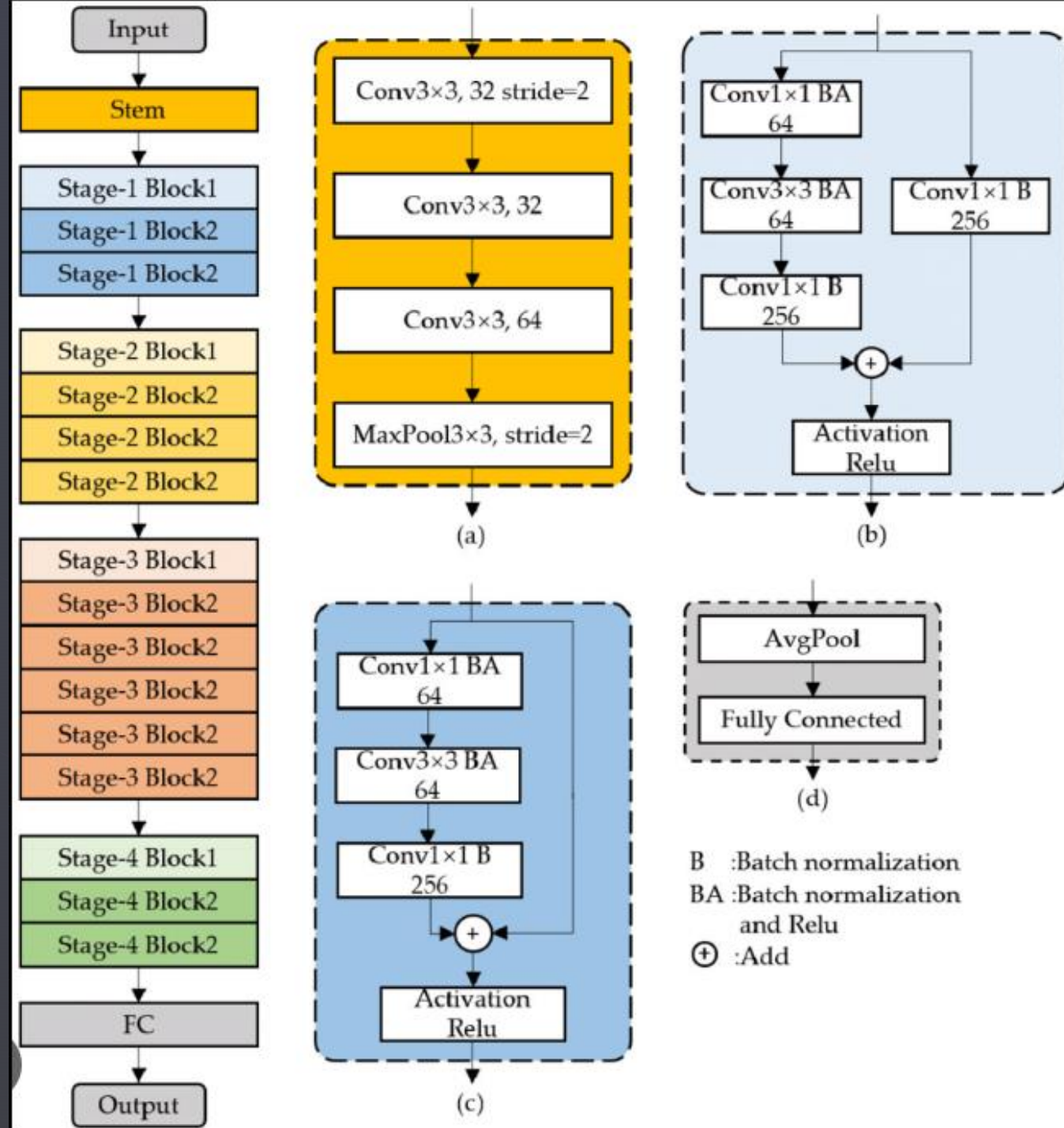
- In Keras, many choices are available, with nearly similar format in code. Some of them are,
 - InceptionV3
 - ResNet50
 - VGG16
 - Xception
- We have chosen only ResNet50 for our model.

How ResNets (residual networks) work

Its major advantage is that it can deal with the vanishing gradient problem

ResNets contains connections different types like,

- a) Simple convolutional layers.
- b) Skip connections (without a Con. Layer in between)
- c) Skip connections (with a Con. Layer in between)
- d) Fully connected layer (in the end)



► How to evaluate the model?

- Differentiating between whether there is DR or not. (A wrong prediction of “DR +ve” is preferred over a wrong prediction of “DR -ve”.)
- Classification accuracy
- Accuracy of differentiating between “no DR”, “non-prolific DR” and “prolific DR”.
- Accuracy for differentiating between the severity of non-prolific DR.

Model: "model_3"

Layer (type)	Output Shape	Param #
=====		
input_8 (InputLayer)	[(None, 128, 128, 3)]	0
sequential_3 (Sequential)	(None, 128, 128, 3)	0
tf.__operators__.getitem_3 (SlicingOpLambda)	(None, 128, 128, 3)	0
tf.nn.bias_add_3 (TFOpLambda)	(None, 128, 128, 3)	0
resnet50 (Functional)	(None, 2048)	23587712
dropout_3 (Dropout)	(None, 2048)	0
dense_3 (Dense)	(None, 5)	10245

=====

Total params: 23,597,957

Trainable params: 10,245

Non-trainable params: 23,587,712

=====

Epoch 9/12

263/263 [=====] - 494s 2s/step - loss: 0.1526 - accuracy: 0.7346 - val_loss: 1.4254 - val_accuracy: 0.7127

Epoch 10/12

263/263 [=====] - 514s 2s/step - loss: 0.1521 - accuracy: 0.7346 - val_loss: 1.3450 - val_accuracy: 0.7127

Epoch 11/12

263/263 [=====] - 522s 2s/step - loss: 0.1494 - accuracy: 0.7348 - val_loss: 1.3790 - val_accuracy: 0.7127

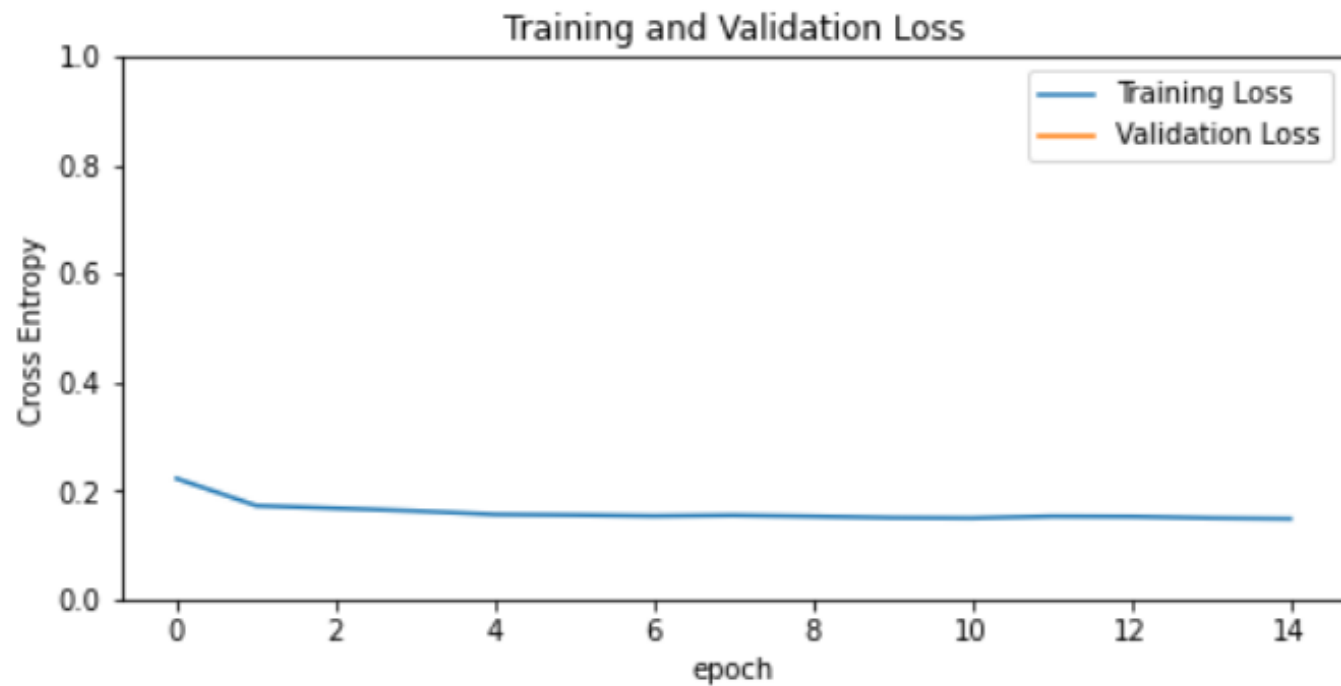
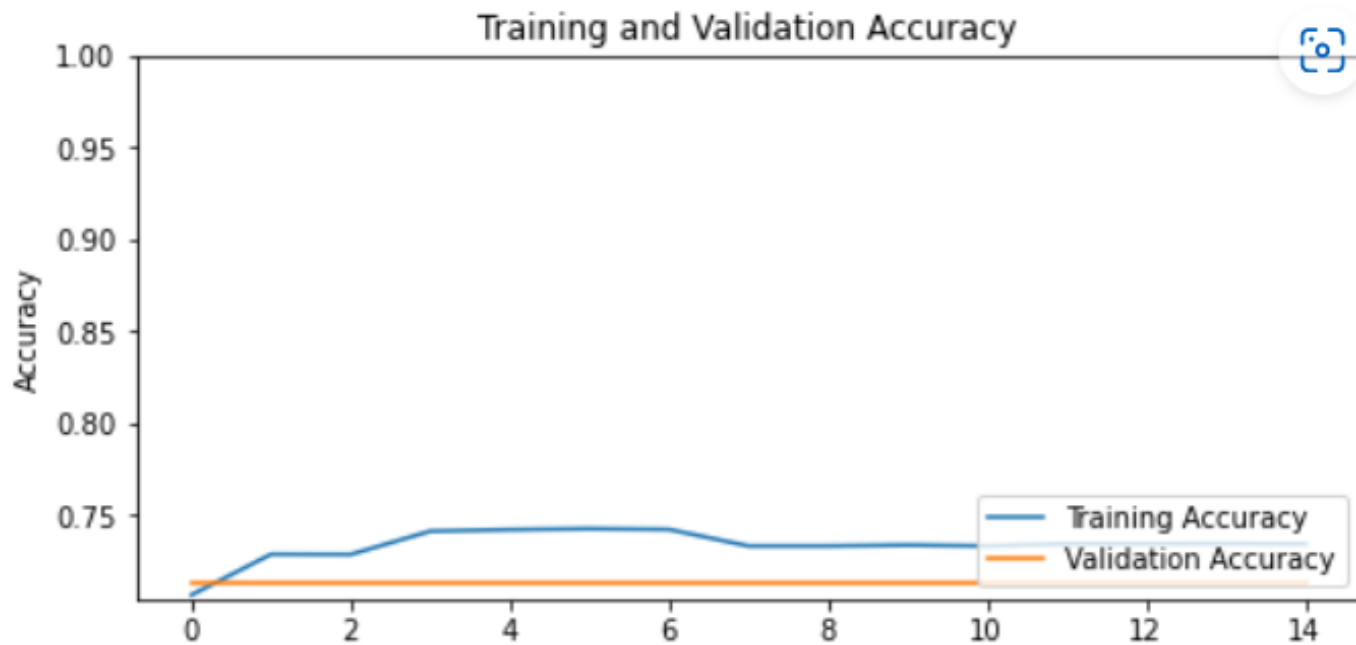
Epoch 12/12

263/263 [=====] - 520s 2s/step - loss: 0.1481 - accuracy: 0.7345 - val_loss: 1.3830 - val_accuracy: 0.7127

- - - - -

Training for train 4 completed

- - - - -




```

prediction
      0  1  2  3  4
actual = 0 1017 0 0 0 0
actual = 1  112 0 0 0 0
actual = 2  230 0 0 0 0
actual = 3   32 0 0 0 0
actual = 4   36 0 0 0 0

```

```

-----
classification report
      precision    recall  f1-score   support

      0       0.71      1.00      0.83      1017
      1       0.00      0.00      0.00       112
      2       0.00      0.00      0.00       230
      3       0.00      0.00      0.00        32
      4       0.00      0.00      0.00        36

   micro avg       0.71      0.71      0.71      1427
   macro avg       0.14      0.20      0.17      1427
weighted avg       0.51      0.71      0.59      1427
 samples avg       0.71      0.71      0.71      1427

```

```

classification accuracy,
0.7126839523475823

```

```

classification between DR and no-DR
1.634197617379117

```

```

classification accuracy for differentiating between prolific and non prolific
0.0

```

```

classification accuracy for differentiating within non-prolific DR
0.0

```

```
# number of layers in the base model,  
print("Number of layers in the base model: ", len(base_model.layers))  
  
# Fine-tune from this layer onwards,  
fine_tune_at = -7 # the number should be -ve  
  
for layer in model.layers[fine_tune_at:]:  
    layer.trainable = True
```

```
Number of layers in the base model: 176
```

Model: "model_3"

Layer (type)	Output Shape	Param #
input_8 (InputLayer)	[(None, 128, 128, 3)]	0
sequential_3 (Sequential)	(None, 128, 128, 3)	0
tf.__operators__.getitem_3 (SlicingOpLambda)	(None, 128, 128, 3)	0
tf.nn.bias_add_3 (TFOpLambda)	(None, 128, 128, 3)	0
resnet50 (Functional)	(None, 2048)	23587712
dropout_3 (Dropout)	(None, 2048)	0
dense_3 (Dense)	(None, 5)	10245

=====
Total params: 23,597,957

Trainable params: 23,544,837

Non-trainable params: 53,120
=====

Epoch 9/12

263/263 [=====] - 1127s 4s/step - loss: 0.1417 - accuracy: 0.7346 - val_loss: 1.3003 - val_accuracy: 0.7127

Epoch 10/12

263/263 [=====] - 1116s 4s/step - loss: 0.1412 - accuracy: 0.7346 - val_loss: 1.3208 - val_accuracy: 0.7127

Epoch 11/12

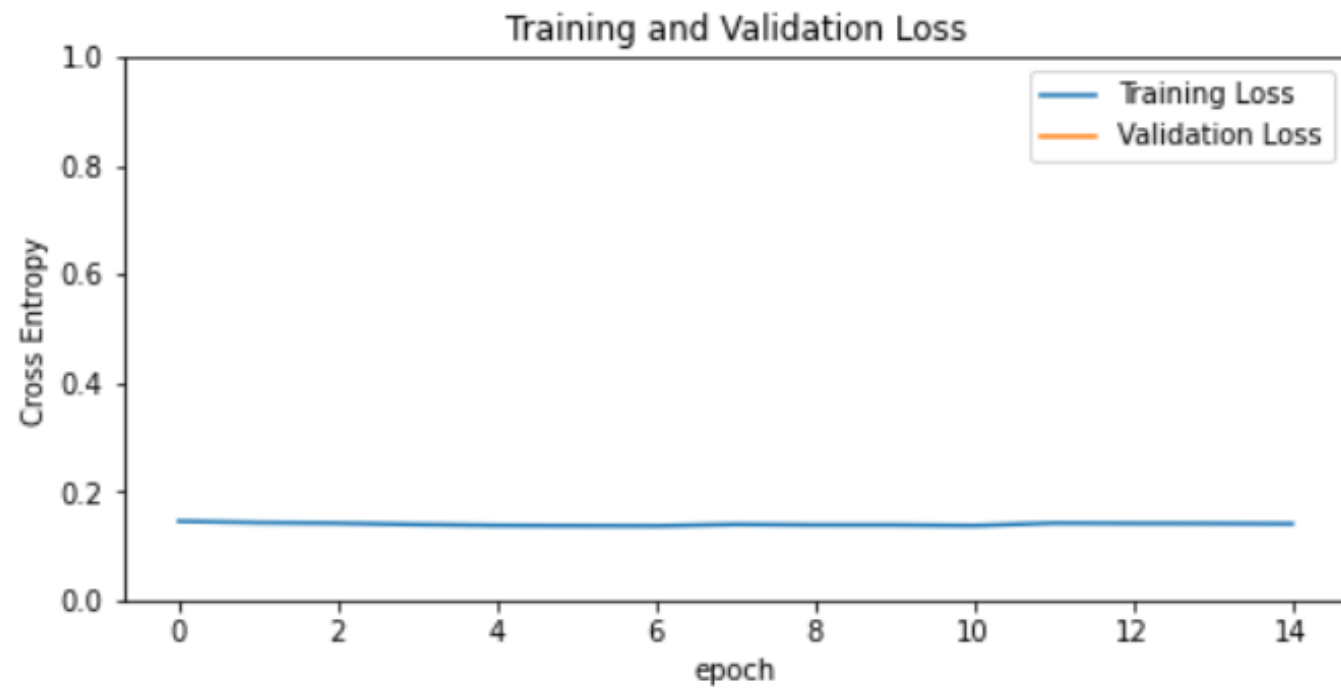
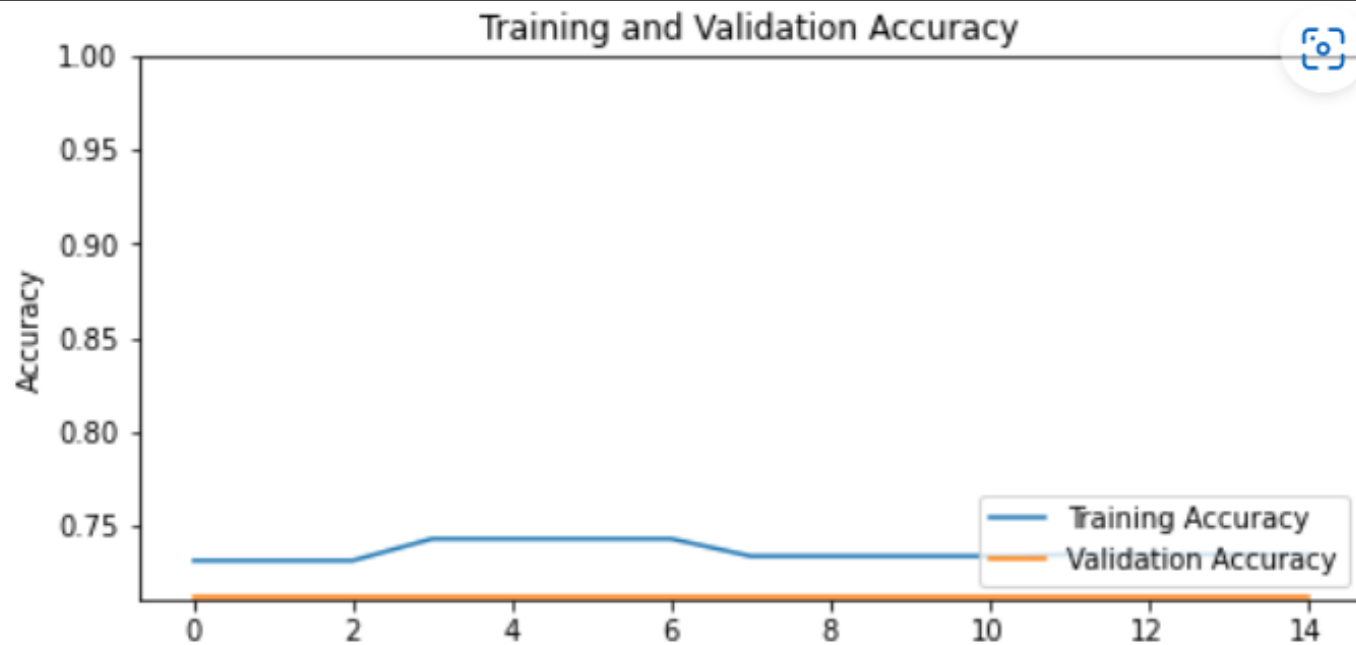
263/263 [=====] - 1409s 5s/step - loss: 0.1410 - accuracy: 0.7346 - val_loss: 1.3054 - val_accuracy: 0.7127

Epoch 12/12

263/263 [=====] - 1472s 6s/step - loss: 0.1404 - accuracy: 0.7346 - val_loss: 1.2626 - val_accuracy: 0.7127

- - - - -
fine tuning for train 4 completed

- - - - -



```

prediction
      0  1  2  3  4
actual = 0 1017 0 0 0 0
actual = 1  112 0 0 0 0
actual = 2  230 0 0 0 0
actual = 3   32 0 0 0 0
actual = 4   36 0 0 0 0

```

classification report

	precision	recall	f1-score	support
0	0.71	1.00	0.83	1017
1	0.00	0.00	0.00	112
2	0.00	0.00	0.00	230
3	0.00	0.00	0.00	32
4	0.00	0.00	0.00	36
micro avg	0.71	0.71	0.71	1427
macro avg	0.14	0.20	0.17	1427
weighted avg	0.51	0.71	0.59	1427
samples avg	0.71	0.71	0.71	1427

classification accuracy,
0.7126839523475823

classification between DR and no-DR
1.634197617379117

classification accuracy for differentiating between prolific and non prolific
0.0

classification accuracy for differentiating within non-prolific DR
0.0

What features is the network trying to find?

What the neural network has learned

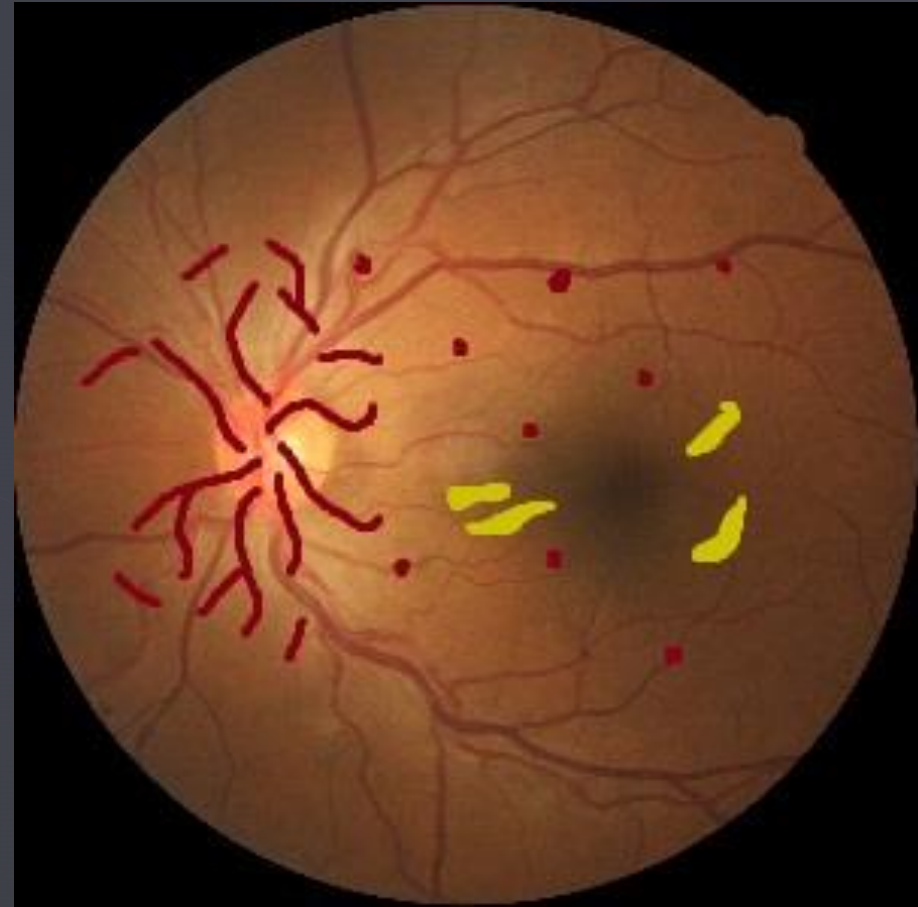
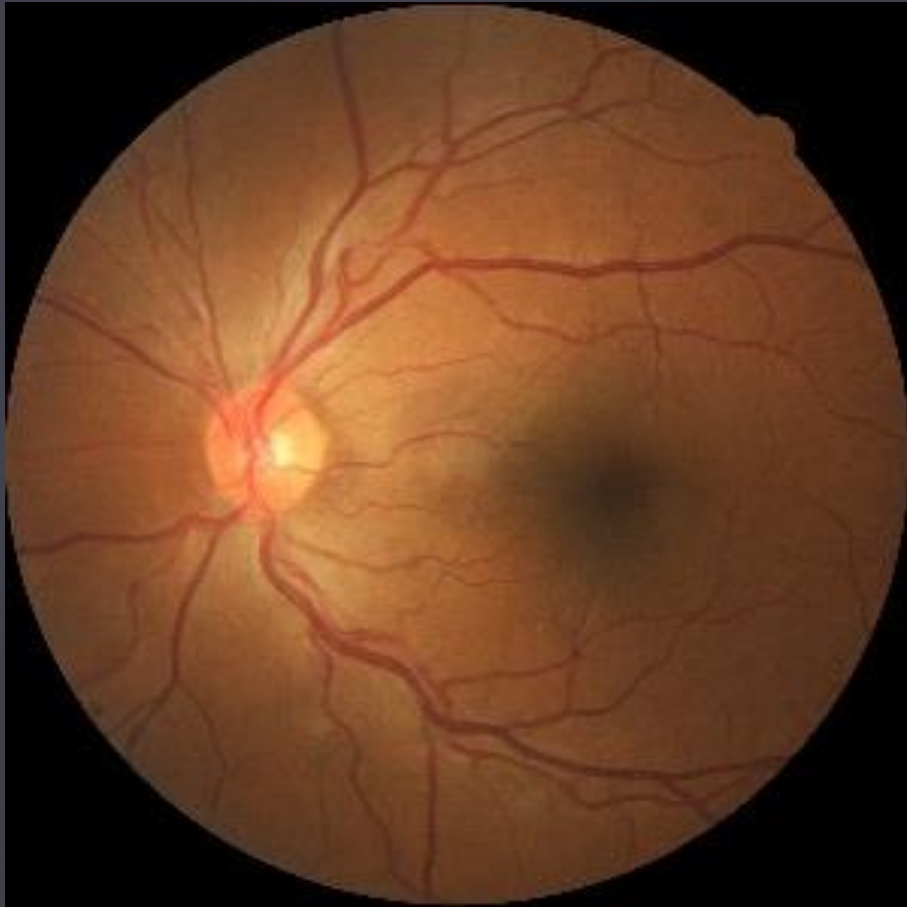
How to approach?

- Weight visualization of filters in the convolutional layers
- By blurring certain areas in DR images (which we think are crucial for the analysis), and then passing the image through the model
- By adding certain features in no-DR images (which we think are crucial for the analysis), and then passing the image through the model
- We can use TensorFlow to get heatmaps which indicate in which parts the model is focusing when doing feature extraction.

In this case, we only use the third approach.

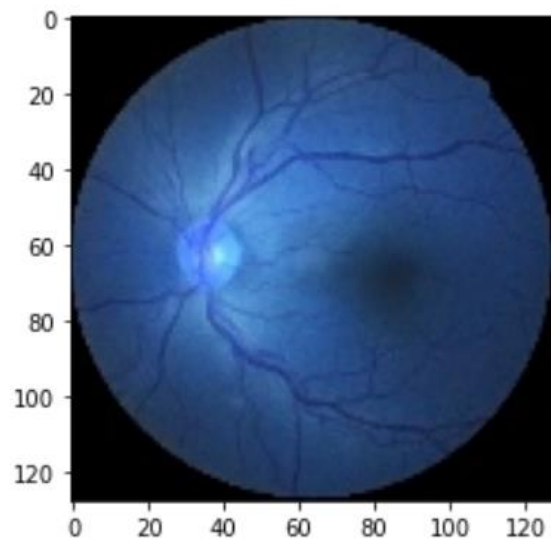
Painted Image analysis

Blood hemorrhages, hard exudates (yellow flakes on retina) and neovascularization were depicted on the no-DR images.

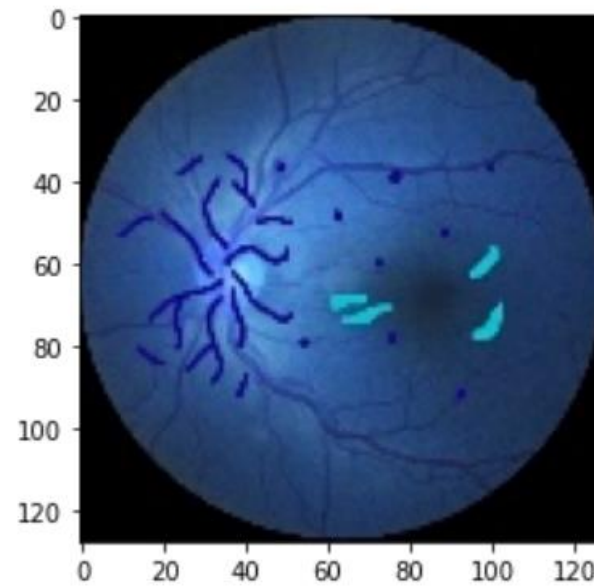


```
-----  
for DR severity, 0  
-----  
image name = 17_left  
level = 0  
new expected level = not 0  
-----
```

true image



```
-----  
new image
```



```
1/1 [=====] - 0s 136ms/step  
predicted level = 0  
-----  
-----
```

Our mistake

- The chosen resolution was too small (an image)
- The batch size was too small (here, 32), compared to the imbalance.
- For small batch sizes, it would have been better to use oversampling or ensemble method, to counter imbalanced data.

Note that, oversampling may be introduced in the program by adding some extra lines in the “images_and_labels” function

```

def images_and_labels ( main_path , name ):
    """
    To take images and their labels, we need,

    (i) main_path
    (ii) folder name

    """
    images = np.array ( [ cv2.imread ( train_path + 'train1\\' + '10_left.jpeg' ) ] )

    for image in os.listdir( main_path + name ) :
        image_array = cv2.imread( main_path + name + '\\' + image )
        images = np.append ( images , [image_array] , axis = 0 )

    images = images [ 1: ]
    """
    for labels, we follow a similar approach
    """
    labels = pd.read_csv ( main_path + 'labels' + name[-1] + '.csv' )
    labels = np.array( labels ['level'] )
    labels = np.append ( labels , 4 )
    labels = to_categorical (labels)
    labels = labels[:-1]

    return images , labels

```




the end

