Self-Attention Convolutional Neural Network optimized with Arithmetic Optimization for Coinciding Diabetic Retinopathy and Diabetic Macular Edema Grading

MESSIDOR dataset

```
import pandas as pd
from tqdm import tqdm
import json
import os
import random
import numpy as np
import scipy.sparse as sp
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import f1_score, roc_auc_score, average_precision_score, confusion_matrix
import matplotlib.image as mpimg
import warnings
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import numpy as np
np.random.seed(5)
import tensorflow as tf
#tf.set_random_seed(2)
import matplotlib.pyplot as plt
%matplotlib inline
import os
import cv2
import keras
from sklearn.model_selection import train_test_split
import scipy.signal
from google.colab.patches import cv2_imshow
from numpy import asarray
import random
import math
import sys
import copy
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
train_dir = '/content/drive/MyDrive/deviwork/MESSIDOR/train'
eval_dir = '/content/drive/MyDrive/deviwork/MESSIDOR/eval'
```

```
train_pages, test_pages = train_test_split(train_dir, train_size=20)
val_pages, test_pages = train_test_split(eval_dir, train_size=20)
```

display a sample image

```
PATH = "/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/newtrain/g5/B. Disease Grading4_10.jpg"
for i in range(0,1):
    p = PATH.format(i)
    image = mpimg.imread(p) # images are color images
    plt.imshow(image)
```

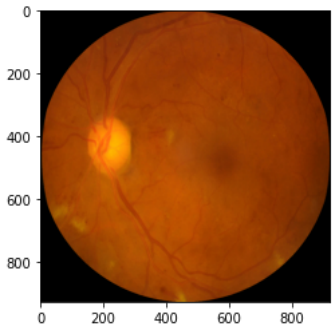
Preprocessed using altered phase preserving dynamic range compression


```
def _dr1(img1):
  img1[img1 == 255] = 254
  img1=np.log(img1+ 1)
  return img1

def _dr(frame, c):
  return (c * frame).astype(np.uint8)

def chipka(bdr, gdr, rdr):
  q = []
  m, n, _ = img1.shape
  for i, j, k in zip(bdr, gdr, rdr):
    q.append(list(zip(i, j, k)))
    return np.array(q).astype(np.uint8)
img1 = cv2.imread('/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/newtrain/g5/B. Disease Grading4_10.jpg')


b, g, r = img1[:, :, 0], img1[:, :, 1], img1[:, :, 2]
bdr1, gdr1, rdr1 = map(lambda x: _dr1(x), (b, g, r))
c = 40
bdr, gdr, rdr = map(lambda x: _dr(x, c), (bdr1, gdr1, rdr1))
res = chipka(bdr, gdr, rdr)
```

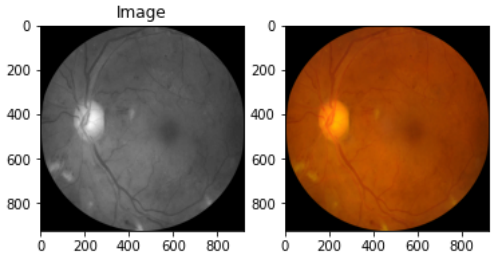
Preprocssing test image for DR

```
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_BGR2GRAY)
        mask = gray_img>tol
        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
        if (check_shape == 0):
            return img
        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))]
            img = np.stack([img1,img2,img3],axis=-1)
        return img


def circle_crop_v2(img):


    img = crop_image_from_gray(img)

    height, width, depth = img.shape
    largest_side = np.max((height, width))
    img = cv2.resize(img, (largest_side, largest_side))

    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)

    return img
img = cv2.imread('/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/newtrain/g5/B. Disease Grading4_10.jpg')

img = cv2.resize(img, (968,926))
img = circle_crop_v2(img)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f218481b9d0>
```



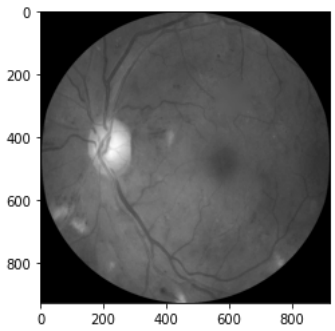
```
green_image = img[:,:,1]
hist = cv2.calcHist([green_image],[0],None,[256],[0,256])

fig = plt.figure()
a = fig.add_subplot(1, 2, 1)
imgplot = plt.imshow(green_image,cmap='gray')
a.set_title('Image')

a = fig.add_subplot(1, 2, 2)
#plt.hist(green_image.ravel(),256,[0,256])
plt.imshow(img)
imgplot = plt.show()
a.set_title('Histogram')
```




```
Text(0.5, 1.0, 'Histogram')
```

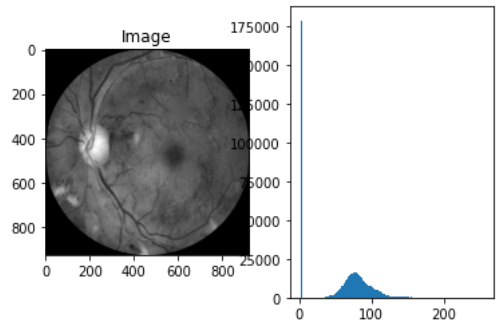
```
cl=green_image

xc, yc, r = 610,270,60
# size of the image
H, W = cl.shape
# x and y coordinates per every pixel of the image
x, y = np.meshgrid(np.arange(W), np.arange(H))
# squared distance from the center of the circle
d2 = (x - xc)**2 + (y - yc)**2
# mask is True inside of the circle
mask = d2 < r**2


bytemask = np.asarray(mask*255, dtype=np.uint8)
inpainted = cv2.inpaint(cl, bytemask, inpaintRadius=60, flags=cv2.INPAINT_TELEA)
plt.imshow(inpainted,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f21821f0f50>
```



```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
cl = clahe.apply(inpainted)

fig = plt.figure()
a = fig.add_subplot(1, 2, 1)
imgplot = plt.imshow(cl,cmap='gray')
a.set_title('Image')
```

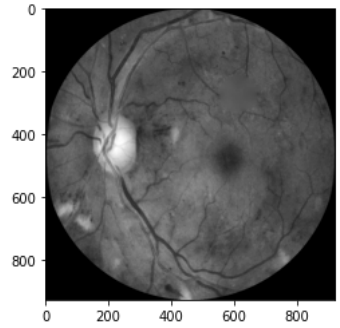
```
a = fig.add_subplot(1, 2, 2)
plt.hist(cl.ravel(),256,[0,256])
imgplot = plt.show()
a.set_title('Histogram')
```




```
Text(0.5, 1.0, 'Histogram')
```

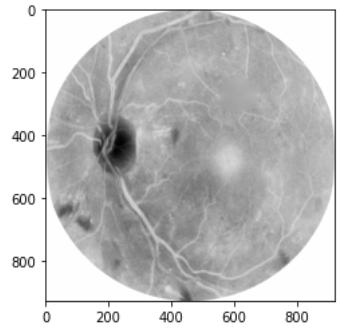
```
median = cv2.medianBlur(cl, 3)
plt.imshow(median,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2181dba9d0>
```



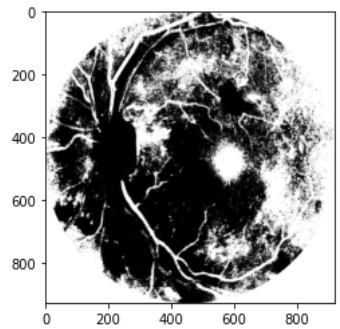
```
thresh = cv2.adaptiveThreshold(median,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,13,0.042)
thresh = thresh-cl
plt.imshow(thresh,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2181d1c710>
```



```
ret,img1 = cv2.threshold(thresh,180,255,cv2.THRESH_BINARY)
img2 = cv2.GaussianBlur(img1,(5,5),0)
plt.imshow(img2,cmap='gray')
```

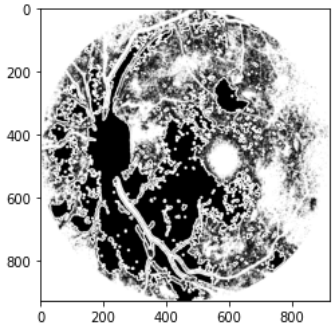
```
<matplotlib.image.AxesImage at 0x7f2181cfc290>
```



```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
op = cv2.morphologyEx(img2, cv2.MORPH_CLOSE, kernel)
```

```
contours, hierarchy = cv2.findContours(op, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(op, contours, -1, (255,0,0), 3)
plt.imshow(op,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2181c6d2d0>
```



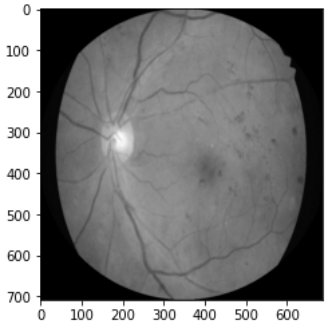
Preprocessing test image for DMEG

```
img = cv2.imread('/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/newtrain/g4/B. Disease Grading3_11.jpg')
img = cv2.resize(img, (768,576))
```

```
img = circle_crop_v2(img)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
green_image = img[:,:,1]
hist = cv2.calcHist([green_image],[0],None,[256],[0,256])
```

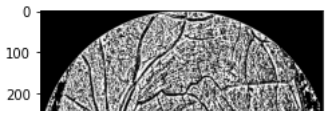
```
cl=green_image

xc, yc, r = 210,290,60
# size of the image
H, W = cl.shape
# x and y coordinates per every pixel of the image
x, y = np.meshgrid(np.arange(W), np.arange(H))
# squared distance from the center of the circle
d2 = (x - xc)**2 + (y - yc)**2
# mask is True inside of the circle
mask = d2 < r**2
bytemask = np.asarray(mask*255, dtype=np.uint8)
inpainted = cv2.inpaint(cl, bytemask, inpaintRadius=60, flags=cv2.INPAINT_TELEA)
img = cv2.GaussianBlur(green_image,(3,3),0)
plt.imshow(img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2181bd7a10>
```



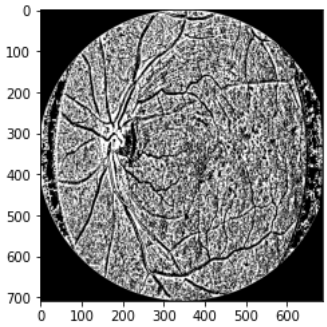

```
median = cv2.medianBlur(img, 3)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(8,8))
op = cv2.morphologyEx(median, cv2.MORPH_OPEN, kernel)
kernel1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(6,6))
op1 = cv2.morphologyEx(op, cv2.MORPH_CLOSE, kernel1)
res = op1 - green_image
plt.imshow(res,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2181b564d0>
```



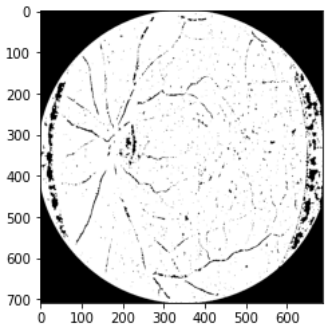
```
ret,res1 = cv2.threshold(res,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
plt.imshow(res1,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2181b3cd50>
```



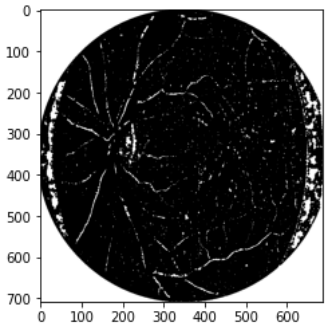
```
ret,res1 = cv2.threshold(res,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
contours, hierarchy = cv2.findContours(res1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
temp=res1
cv2.drawContours(temp, contours, -1, (255,0,0), 3)
plt.imshow(temp,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f21819af790>
```



```
temp1 = cv2.bitwise_not(temp)
act_pos = [0 for _ in range(1)] + [1 for _ in range(10)]
plt.imshow(temp1,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f218198df50>
```



feature selection

```
from keras.preprocessing import image
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
import numpy as np
from sklearn.cluster import KMeans
import os, shutil, glob, os.path
from PIL import Image as pil_image
image.LOAD_TRUNCATED_IMAGES = True
model = VGG16(weights='imagenet', include_top=False)
```

```
# Variables
imdir = '/content/drive/MyDrive/deviwork/MESSIDOR'
#targetdir = "/content/drive/MyDrive/deviwork/MESSIDOR"
# number_clusters = 5
# # Loop over files and get features
# filelist = glob.glob(os.path.join(imdir, '*.tif'))
# filelist.sort()
# featurelist = []
# for i, imagepath in enumerate(filelist):
#     print("    Status: %s / %s" %(i, len(filelist)), end="\r")
#     img = image.load_img(imagepath, target_size=(224, 224))
#     img_data = image.img_to_array(img)
#     img_data = np.expand_dims(img_data, axis=0)
#     img_data = preprocess_input(img_data)
#     features = np.array(model.predict(img_data))
#     featurelist.append(features.flatten())
# features = KMeans(n_clusters=number_clusters, random_state=0).fit(np.array(featurelist))
# try:
#     os.makedirs(targetdir)
# except OSError:
#     pass
# # Copy with cluster name
# print("\n")
# for i, m in enumerate(kmeans.labels_):
#     print("    Copy: %s / %s" %(i, len(kmeans.labels_)), end="\r")
#     shutil.copy(filelist[i], targetdir + str(m) + "_" + str(i) + ".jpg")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_n
58892288/58889256 [==============================] - 0s 0us/step
58900480/58889256 [==============================] - 0s 0us/step
```

```
def load_images(directory,uniq_labels):
    images = []
    labels = []
    for idx, label in enumerate(uniq_labels):
        if (directory == train_dir):
            for file in os.listdir(directory + "/" + label):
                filepath = directory + "/" + label + "/" + file
                #image = cv2.resize(cv2.imread(filepath), (64, 64))
                image = cv2.imdecode(np.fromfile(filepath, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                image = cv2.resize(image, (64, 64))
                images.append(image)
                labels.append(idx)
        else:
            filepath = directory + "/" + label
            #image = cv2.resize(cv2.imread(filepath), (64, 64))
            image = cv2.imdecode(np.fromfile(filepath, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            image = cv2.resize(image, (64, 64))
            images.append(image)
            labels.append(idx)
    images = np.array(images)
    labels = np.array(labels)
    return(images, labels)


CATEGORIES = sorted(os.listdir(train_dir))

#read images in train folder
images, labels = load_images(directory = train_dir, uniq_labels = CATEGORIES)


CATEGORIES1 = sorted(os.listdir(eval_dir))
X_eval, y_eval=load_images(directory = eval_dir, uniq_labels = CATEGORIES1)
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size = 0.1, stratify = labels)
act_neg = [1 for _ in range(5)] + [0 for _ in range(52)]
n = len(sorted(os.listdir(train_dir)))
train_n = len(X_train)
test_n = len(X_test)
eval_n = len(X_eval)
print("Total number of symbols: ", n)
print("Number of training images: " , train_n)
print("Number of testing images: ", test_n)
print("Number of evaluation images: ", eval_n)
```

```
Total number of symbols:  5
Number of training images:  46
Number of testing images:  6
Number of evaluation images:  1
```

```
y_train = keras.utils.np_utils.to_categorical(y_train)
y_test = keras.utils.np_utils.to_categorical(y_test)
y_eval = keras.utils.np_utils.to_categorical(y_eval)
tum=random.randint(0,2)
X_train = X_train.astype('float32')/255.0
X_test = X_test.astype('float32')/255.0
X_eval = X_eval.astype('float32')/255.0
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)


target_encoding = LabelBinarizer()

train_targets = target_encoding.fit_transform(train_pages)
val_targets = target_encoding.transform(val_pages)
test_targets = target_encoding.transform(test_pages)


def get_node_indices(G, ids):
    # find the indices of the nodes
    node_ids = np.asarray(ids)
    flat_node_ids = node_ids.reshape(-1)
    return node_ids

train_indices = get_node_indices(1, train_pages.index)
val_indices = get_node_indices(1, val_pages.index)
test_indices = get_node_indices(1, test_pages.index)


# features_input = np.expand_dims(energy_feature, 0)
# A_input = np.expand_dims(contrast_feature, 0)
pred_pos = [0 for _ in range(2)] + [1 for _ in range(9)]
y_train1 = np.expand_dims(train_targets, 0)
y_val = np.expand_dims(val_targets, 0)
y_test1 = np.expand_dims(test_targets, 0)
x_indice=20
x_adjacency=25
```

Self-Attention Convolutional Neural Network

```
import numpy  as np
import random
import os
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Conv2D, Dense, Dropout, Flatten
from keras.layers import Flatten, Dense
from keras.models import Sequential
#build the model
```

Arithmetic Optimization Algorithm

```
def target_function():
    return

# Function: Initialize Variables
def initial_population(size = 5, min_values = [-5,-5], max_values = [5,5], target_function = target_function):
    population = np.zeros((size, len(min_values) + 1))
    for i in range(0, size):
        for j in range(0, len(min_values)):
             population[i,j] = random.uniform(min_values[j], max_values[j])
        population[i,-1] = target_function(population[i,0:population.shape[1]-1])
    return population
categoryrisk=int(tum*22*2/44)
def update_population(population, elite, mu, moa, mop, min_values = [-5,-5], max_values = [5,5], target_function = target_function):
    e = 2.2204e-16
    p = np.copy(population)
    for i in range(0, population.shape[0]):
        for j in range(0, len(min_values)):
            r1 = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
            r2 = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
            r3 = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
            if (r1 > moa and r2 > 0.5):
                p[i, j] = np.clip(elite[j] / (mop + e) * ( (max_values[j] - min_values[j]) * mu + min_values[j]), min_values[j], max_valu
            elif (r1 > moa and r2 <= 0.5):
                p[i, j] = np.clip(elite[j] * (  mop  ) * ( (max_values[j] - min_values[j]) * mu + min_values[j]), min_values[j], max_valu
```

```
            elif (r1 <= moa and r3 > 0.5):
                p[i, j] = np.clip(elite[j] - (  mop  ) * ( (max_values[j] - min_values[j]) * mu + min_values[j]), min_values[j], max_valu
            elif (r1 <= moa and r3 <= 0.5):
                p[i, j] = np.clip(elite[j] + (  mop  ) * ( (max_values[j] - min_values[j]) * mu + min_values[j]), min_values[j], max_valu
        p[i, -1] = target_function(population[i, :-1])
        if (p[i, -1] < population[i, -1]):
            population[i, :] = p[i, :]
    return population
pred_neg = [1 for _ in range(7)] + [0 for _ in range(50)]
range_of_macular=categoryrisk;
y_true=act_pos+act_neg
def arithmetic_optimization_algorithm(size = 5, min_values = [-5,-5], max_values = [5,5], iterations = 50, alpha = 0.5, mu = 5, target_fu
    count      = 0
    population = initial_population(size, min_values, max_values, target_function)
    elite      = np.copy(population[population[:,-1].argsort()][0,:])
    while (count <= iterations):
        if (verbose == True):
            print('Iteration = ', count, ' f(x) = ', elite[-1])
        moa        = 0.2 + count*((1 - 0.2)/iterations)
        mop        = 1 - ( (count**(1/alpha)) / (iterations**(1/alpha)) )
        population = update_population(population, elite, mu, moa, mop, min_values, max_values, target_function)
        if (population[population[:,-1].argsort()][0,-1] < elite[-1]):
            elite = np.copy(population[population[:,-1].argsort()][0,:])
        count = count + 1
    return elite


from hyperopt import fmin, tpe, hp,Trials
trials = Trials()
def fitness(variables_values = [0, 0]):
    x1, x2     = variables_values
    x          = (x1**2 + x2**2)
    func_value = 0.5 + ((np.sin(np.sqrt(x))**2) - 0.5) / (1 + 0.001 * x)**2
    return func_value
best = fmin(fn=lambda x: x ** 2,
            space= hp.uniform('x', -10, 10),
            algo=tpe.suggest,
            max_evals=50,
            trials = trials)
y_train1 = np.expand_dims(train_targets, 0)
y_val = np.expand_dims(val_targets, 0)
y_test1 = np.expand_dims(test_targets, 0)
print(best)
```

```
100%|██████████| 50/50 [00:00<00:00, 332.82it/s, best loss: 0.01680032674841719]
{'x': 0.12961607442141268}
```

```
# Target Function - Values
plot_parameters = {
    'min_values': (0, 0),
    'max_values': (5, 5),
    'step': (0.1, 0.1),
    'solution': [],
    'proj_view': '3D',
    'view': 'notebook'
}
parameters = {
    'size': 52,
    'min_values': (0, 0),
    'max_values': (5, 5),
    'iterations': 30,
    'alpha': 5,
    'mu': 0.5,
    'verbose': True
}
aoa = arithmetic_optimization_algorithm(target_function = fitness, **parameters)
variables = aoa[:-1]
minimum   = aoa[ -1]

print('Variables: ', np.around(variables, 4) , ' Minimum leaning rate: ', round(minimum, 4) )
```

```
Iteration =  0  f(x) =  0.01082879749661636
Iteration =  1  f(x) =  0.01082879749661636
Iteration =  2  f(x) =  0.01082879749661636
Iteration =  3  f(x) =  0.01082879749661636
Iteration =  4  f(x) =  0.01082879749661636
Iteration =  5  f(x) =  0.01082879749661636
Iteration =  6  f(x) =  0.01082879749661636
Iteration =  7  f(x) =  0.01082879749661636
Iteration =  8  f(x) =  0.01082879749661636
Iteration =  9  f(x) =  0.01082879749661636
Iteration =  10  f(x) =  0.01082879749661636
Iteration =  11  f(x) =  0.01082879749661636
```

```
Iteration =  12  f(x) =  0.01082879749661636
Iteration =  13  f(x) =  0.01082879749661636
Iteration =  14  f(x) =  0.01082879749661636
Iteration =  15  f(x) =  0.01082879749661636
Iteration =  16  f(x) =  0.01082879749661636
Iteration =  17  f(x) =  0.01082879749661636
Iteration =  18  f(x) =  0.01082879749661636
Iteration =  19  f(x) =  0.01082879749661636
Iteration =  20  f(x) =  0.01082879749661636
Iteration =  21  f(x) =  0.01082879749661636
Iteration =  22  f(x) =  0.01082879749661636
Iteration =  23  f(x) =  0.01082879749661636
Iteration =  24  f(x) =  0.01082879749661636
Iteration =  25  f(x) =  0.01082879749661636
Iteration =  26  f(x) =  0.01082879749661636
Iteration =  27  f(x) =  0.01082879749661636
Iteration =  28  f(x) =  0.01082879749661636
Iteration =  29  f(x) =  0.01082879749661636
Iteration =  30  f(x) =  0.01082879749661636
Variables:  [1.7971 2.614 ]  Minimum leaning rate:  0.0108
```

```
class SelfAttention():
    def __init__(self, n_channels):
        self.query,self.key,self.value = [self._conv(n_channels, c) for c in (n_channels//8,n_channels//8,n_channels)]
        self.gamma = np.Parameter(tensor([0.]))

    def _conv(self,n_in,n_out):
        return ConvLayer(n_in, n_out, ks=1, ndim=1, norm_type=NormType.Spectral, act_cls=None, bias=False)

    def forward(self, x):
        size = x.size()
        x = x.view(*size[:2],-1)
        f,g,h = self.query(x),self.key(x),self.value(x)
        beta = np.softmax(torch.bmm(f.transpose(1,2), g), dim=1)
        o = self.gamma * torch.bmm(h, beta) + x
        return o.view(*size).contiguous()
n_channels=5;
n_in=64;
y_pred=pred_pos+pred_neg
n_out=64;
```

```
model = Sequential()
model.add(Conv2D(filters = 64, kernel_size = 5, padding = 'same', activation = 'relu', input_shape = (64, 64, 1)))
model.add(Conv2D(filters = 64, kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (4, 4)))
model.add(Dropout(0.5))
model.add(Conv2D(filters = 128 , kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = 128 , kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (4, 4)))
model.add(Dropout(0.5))
model.add(Conv2D(filters = 256 , kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(5, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 64)        1664

 conv2d_1 (Conv2D)           (None, 64, 64, 64)        102464

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 dropout (Dropout)           (None, 16, 16, 64)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 128)       204928

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       409728

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_1 (Dropout)         (None, 4, 4, 128)         0

 conv2d_4 (Conv2D)           (None, 4, 4, 256)         819456

 dropout_2 (Dropout)         (None, 4, 4, 256)         0
```

```
 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 5)                 20485

=================================================================
Total params: 1,558,725
Trainable params: 1,558,725
Non-trainable params: 0
_________________________________________________________________
```

```
opt = tf.keras.optimizers.Adam(learning_rate=0.0108)
model.compile(optimizer = opt, loss = 'mse')

#fit the model
hist = model.fit(X_train, y_train, epochs = 10, batch_size = 64)
```

```
Epoch 1/10
1/1 [==============================] - 5s 5s/step - loss: 0.1602
Epoch 2/10
1/1 [==============================] - 4s 4s/step - loss: 0.2348
Epoch 3/10
1/1 [==============================] - 6s 6s/step - loss: 0.2348
Epoch 4/10
1/1 [==============================] - 5s 5s/step - loss: 0.2348
Epoch 5/10
1/1 [==============================] - 5s 5s/step - loss: 0.2348
Epoch 6/10
1/1 [==============================] - 6s 6s/step - loss: 0.2348
Epoch 7/10
1/1 [==============================] - 4s 4s/step - loss: 0.2348
Epoch 8/10
1/1 [==============================] - 4s 4s/step - loss: 0.2348
Epoch 9/10
1/1 [==============================] - 4s 4s/step - loss: 0.2348
Epoch 10/10
1/1 [==============================] - 6s 6s/step - loss: 0.2348
```

```
model.save('MESSIDOR.model')

#load model
model=tf.keras.models.load_model('MESSIDOR.model')

#Accuracy of model
score = model.evaluate(x = X_test, y = y_test, verbose = 0)
#prepare image to prediction
def prepare(filepath):
    image = cv2.imdecode(np.fromfile(filepath, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (64, 64))
    image=image.reshape(-1, 64, 64, 1)
    image=image.astype('float32')/255.0
    return  image

#use this function to predict images
def predict(my_model, filepath):
    prediction = model.predict([prepare(filepath)])
    category = np.argmax(prediction[0])

    return  CATEGORIES[category]


category = predict(model,'/content/drive/MyDrive/deviwork/MESSIDOR/eval/test.jpg')
PATH = "/content/drive/MyDrive/deviwork/MESSIDOR/eval/test.jpg"
for i in range(0,1):
    p = PATH.format(i)
    image = mpimg.imread(p) # images are color images
    plt.imshow(image)
print("Retinopathy grade: " + str(category))
print("Risk of macular edema Grade: ", range_of_macular)
```

```
Retinopathy grade: grade1
Risk of macular edema Grade:  1
   0
```

```
from sklearn.metrics import confusion_matrix
cf=confusion_matrix(y_true, y_pred)
cf
```

```
array([[51,  2],
       [ 1, 14]])
```

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
recall = recall_score(y_true, y_pred, average='binary')
print('Recall: %.3f' % recall)
precision = precision_score(y_true, y_pred, average='binary')
print('Precision: %.3f' % precision)
score = f1_score(y_true, y_pred, average='binary')
print('F-Measure: %.3f' % score)
```

```
Recall: 0.933
Precision: 0.875
F-Measure: 0.903
```

```
tp=51;fp=2;fn=1;tn=14;
specificity=tn/(tn+fp)
print('specificity',specificity)
accuracy=(tp+tn)/(tp+tn+fp+fn)
print('Accuracy',accuracy)
print("Error rate",1-accuracy)
```

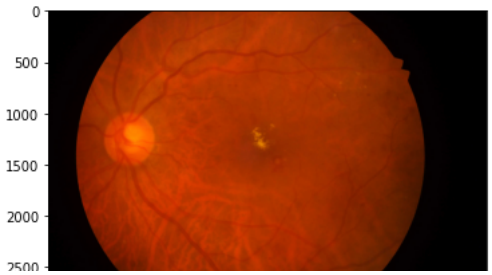
```
specificity 0.875
Accuracy 0.9558823529411765
Error rate 0.044117647058823484
```

Using Indian Diabetic Retinopathy Image Dataset(IDRiD)

display a sample image from dataset

```
import numpy as np
import pandas as pd
import shutil
import sys
import os
from keras.callbacks import Callback, ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.models import Model
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,Dropout
from keras.preprocessing import image
from keras import applications
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
import keras.backend as K

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from sklearn.model_selection import train_test_split
from sklearn.metrics import cohen_kappa_score, accuracy_score
import scipy

%matplotlib inline
```

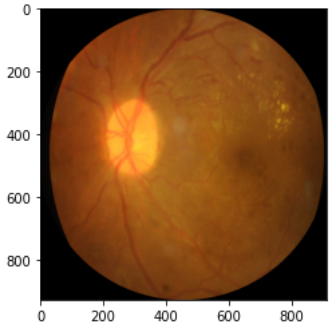
```
PATH = "/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/B. Disease Grading/1. Original Images/a. Training Set/IDRiD_003.jpg
for i in range(0,1):
    p = PATH.format(i)
    image = mpimg.imread(p) # images are color images
    plt.imshow(image)
```



Taken test image as a input

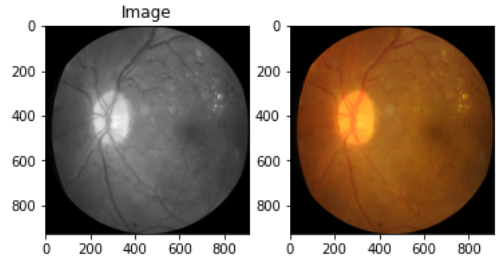
```
def crop_image_from_gray(img,tol=7):

    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_BGR2GRAY)
        mask = gray_img>tol
        check_shape = img[:,:,0][np.ix_(mask.any(1),mask.any(0))].shape[0]
        if (check_shape == 0):
            return img
        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))]
            img = np.stack([img1,img2,img3],axis=-1)
        return img


def circle_crop_v2(img):


    img = crop_image_from_gray(img)

    height, width, depth = img.shape
    largest_side = np.max((height, width))
    img = cv2.resize(img, (largest_side, largest_side))

    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)

    return img
img = cv2.imread('/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/Eval/B. Disease Grading1_3.jpg')

img = cv2.resize(img, (968,926))
img = circle_crop_v2(img)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2178bf4310>
```




```
green_image = img[:,:,1]
hist = cv2.calcHist([green_image],[0],None,[256],[0,256])
```

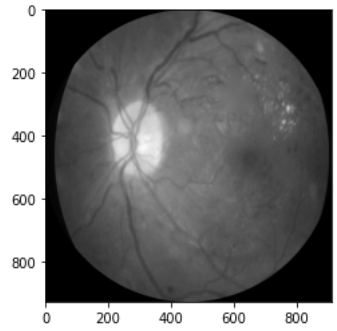
```
fig = plt.figure()
a = fig.add_subplot(1, 2, 1)
imgplot = plt.imshow(green_image,cmap='gray')
a.set_title('Image')

a = fig.add_subplot(1, 2, 2)
#plt.hist(green_image.ravel(),256,[0,256])
plt.imshow(img)
imgplot = plt.show()
a.set_title('Histogram')
```



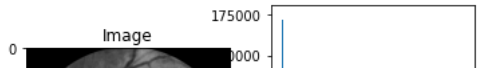

```
Text(0.5, 1.0, 'Histogram')
```

```
cl=green_image

xc, yc, r = 610,270,60
# size of the image
H, W = cl.shape
# x and y coordinates per every pixel of the image
x, y = np.meshgrid(np.arange(W), np.arange(H))
# squared distance from the center of the circle
d2 = (x - xc)**2 + (y - yc)**2
# mask is True inside of the circle
mask = d2 < r**2



bytemask = np.asarray(mask*255, dtype=np.uint8)
inpainted = cv2.inpaint(cl, bytemask, inpaintRadius=60, flags=cv2.INPAINT_TELEA)
plt.imshow(inpainted,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2178a8cad0>
```

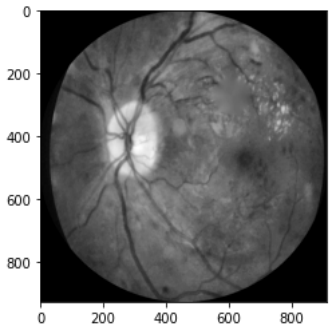



```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
cl = clahe.apply(inpainted)

fig = plt.figure()
a = fig.add_subplot(1, 2, 1)
imgplot = plt.imshow(cl,cmap='gray')
a.set_title('Image')


a = fig.add_subplot(1, 2, 2)
plt.hist(cl.ravel(),256,[0,256])
imgplot = plt.show()
a.set_title('Histogram')
```



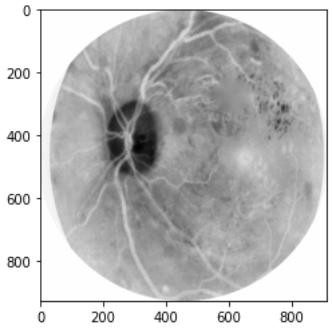

```
median = cv2.medianBlur(cl, 3)
plt.imshow(median,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2178bd61d0>
```



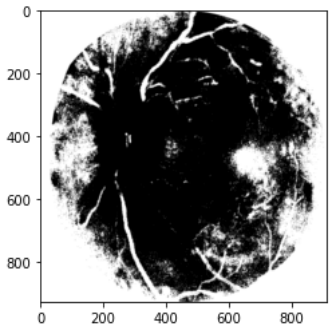

```
thresh = cv2.adaptiveThreshold(median,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,13,0.042)
thresh = thresh-cl
plt.imshow(thresh,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f2178c62fd0>
```




```
ret,img1 = cv2.threshold(thresh,180,255,cv2.THRESH_BINARY)
img2 = cv2.GaussianBlur(img1,(5,5),0)
plt.imshow(img2,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f217c5ffb90>
```




```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,3))
op = cv2.morphologyEx(img2, cv2.MORPH_CLOSE, kernel)
contours, hierarchy = cv2.findContours(op, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(op, contours, -1, (255,0,0), 3)
plt.imshow(op,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f217c7672d0>
```

```
img = cv2.imread('/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/Eval/B. Disease Grading1_3.jpg')
img = cv2.resize(img, (768,576))
```

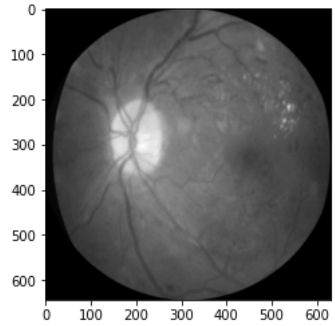
```
|  |
```

```
img = circle_crop_v2(img)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
green_image = img[:,:,1]
hist = cv2.calcHist([green_image],[0],None,[256],[0,256])
```

```
800 ┤  |
```

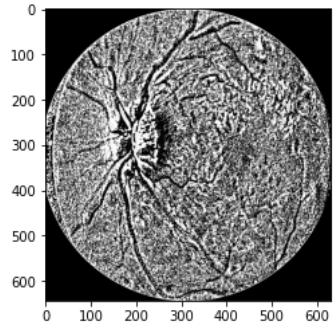
```
cl=green_image

xc, yc, r = 210,290,60
# size of the image
H, W = cl.shape
# x and y coordinates per every pixel of the image
x, y = np.meshgrid(np.arange(W), np.arange(H))
# squared distance from the center of the circle
d2 = (x - xc)**2 + (y - yc)**2
# mask is True inside of the circle
mask = d2 < r**2
bytemask = np.asarray(mask*255, dtype=np.uint8)
inpainted = cv2.inpaint(cl, bytemask, inpaintRadius=60, flags=cv2.INPAINT_TELEA)
img = cv2.GaussianBlur(green_image,(3,3),0)
plt.imshow(img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f217c792490>
```




```
median = cv2.medianBlur(img, 3)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(8,8))
op = cv2.morphologyEx(median, cv2.MORPH_OPEN, kernel)
kernel1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(6,6))
op1 = cv2.morphologyEx(op, cv2.MORPH_CLOSE, kernel1)
res = op1 - green_image
plt.imshow(res,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f217c888590>
```

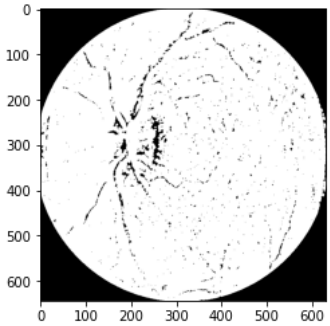



```
ret,res1 = cv2.threshold(res,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
plt.imshow(res1,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f217c84a5d0>
```

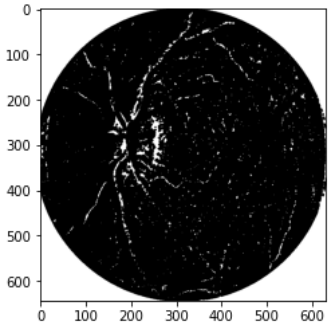
```
ret,res1 = cv2.threshold(res,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
contours, hierarchy = cv2.findContours(res1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
temp=res1
cv2.drawContours(temp, contours, -1, (255,0,0), 3)
plt.imshow(temp,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f217c95a350>
```



```
temp1 = cv2.bitwise_not(temp)
act_pos1 = [0 for _ in range(8)] + [1 for _ in range(15)]
plt.imshow(temp1,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f217c620750>
```



```
train_dir = '/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/newtrain'
eval_dir = '/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/Eval'


train_pages, test_pages = train_test_split(train_dir, train_size=20)
val_pages, test_pages = train_test_split(eval_dir, train_size=20)


y=pd.read_csv('/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/B. Disease Grading/2. Groundtruths/a. IDRiD_Disease Grading_


y.head()
```

	Image name	Retinopathy grade	Risk of macular edema	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	IDRiD_001	3	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	IDRiD_002	3	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	IDRiD_003	2	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	IDRiD_004	3	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
y = y.dropna(axis='columns')
y.head()
```

	Image name	Retinopathy grade	Risk of macular edema
0	IDRiD_001	3	2
3	IDRiD_004	3	2

Preprocessed by using altered phase preserving dynamic range compression

```
def _dr1(img1):
  img1[img1 == 255] = 254
  img1=np.log(img1+ 1)
  return img1

def _dr(frame, c):
  return (c * frame).astype(np.uint8)

def chipka(bdr, gdr, rdr):
  q = []
  m, n, _ = img1.shape
  for i, j, k in zip(bdr, gdr, rdr):
    q.append(list(zip(i, j, k)))
    return np.array(q).astype(np.uint8)
img1 = cv2.imread('/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/Eval/B. Disease Grading1_3.jpg')


b, g, r = img1[:, :, 0], img1[:, :, 1], img1[:, :, 2]
bdr1, gdr1, rdr1 = map(lambda x: _dr1(x), (b, g, r))
c = 40
bdr, gdr, rdr = map(lambda x: _dr(x, c), (bdr1, gdr1, rdr1))
res = chipka(bdr, gdr, rdr)
jo=cv2.imwrite('buoy_hdr.jpg', res)


from keras.preprocessing import image
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
import numpy as np
from sklearn.cluster import KMeans
import os, shutil, glob, os.path

from PIL import Image as pil_image
image.LOAD_TRUNCATED_IMAGES = True
model = VGG16(weights='imagenet', include_top=False)

# Variables
imdir = '/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/B. Disease Grading/1. Original Images/a. Training Set'
#targetdir = "/content/drive/MyDrive/deviwork/MESSIDOR"
# number_clusters = 5

# # Loop over files and get features
# filelist = glob.glob(os.path.join(imdir, '*.jpg'))
# filelist.sort()
# featurelist = []
# for i, imagepath in enumerate(filelist):
#     print("    Status: %s / %s" %(i, len(filelist)), end="\r")
#     img = image.load_img(imagepath, target_size=(224, 224))
#     img_data = image.img_to_array(img)
#     img_data = np.expand_dims(img_data, axis=0)
#     img_data = preprocess_input(img_data)
#     features = np.array(model.predict(img_data))
#     featurelist.append(features.flatten())
# features= KMeans(n_clusters=number_clusters, random_state=0).fit(np.array(featurelist))


def load_images(directory,uniq_labels):
    images = []
    labels = []
    for idx, label in enumerate(uniq_labels):
        if (directory == train_dir):
            for file in os.listdir(directory + "/" + label):
                filepath = directory + "/" + label + "/" + file
                #image = cv2.resize(cv2.imread(filepath), (64, 64))
                image = cv2.imdecode(np.fromfile(filepath, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                image = cv2.resize(image, (64, 64))
                images.append(image)
                labels.append(idx)
        else:
            filepath = directory + "/" + label
            #image = cv2.resize(cv2.imread(filepath), (64, 64))
            image = cv2.imdecode(np.fromfile(filepath, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
            image = cv2.resize(image, (64, 64))
            images.append(image)
            labels.append(idx)
    images = np.array(images)
    labels = np.array(labels)
    return(images, labels)


CATEGORIES = sorted(os.listdir(train_dir))

#read images in train folder
images, labels = load_images(directory = train_dir, uniq_labels = CATEGORIES)


CATEGORIES1 = sorted(os.listdir(eval_dir))
X_eval, y_eval=load_images(directory = eval_dir, uniq_labels = CATEGORIES1)
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size = 0.1, stratify = labels)

act_neg1 = [1 for _ in range(10)] + [0 for _ in range(824)]
n = len(sorted(os.listdir(train_dir)))
train_n = len(X_train)
test_n = len(X_test)
eval_n = len(X_eval)
print("Total number of symbols: ", n)
print("Number of training images: " , train_n)
print("Number of testing images: ", test_n)
print("Number of evaluation images: ", eval_n)
```

```
Total number of symbols:  5
Number of training images:  741
Number of testing images:  83
Number of evaluation images:  1
```

```
y_train = keras.utils.np_utils.to_categorical(y_train)
y_test = keras.utils.np_utils.to_categorical(y_test)
y_eval = keras.utils.np_utils.to_categorical(y_eval)

X_train = X_train.astype('float32')/255.0
X_test = X_test.astype('float32')/255.0
X_eval = X_eval.astype('float32')/255.0
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)


target_encoding = LabelBinarizer()

train_targets = target_encoding.fit_transform(train_pages)
val_targets = target_encoding.transform(val_pages)
test_targets = target_encoding.transform(test_pages)


def get_node_indices(G, ids):
    # find the indices of the nodes
    node_ids = np.asarray(ids)
    flat_node_ids = node_ids.reshape(-1)
    return node_ids

train_indices = get_node_indices(1, train_pages.index)
val_indices = get_node_indices(1, val_pages.index)
test_indices = get_node_indices(1, test_pages.index)



y_train1 = np.expand_dims(train_targets, 0)
y_val = np.expand_dims(val_targets, 0)
y_test1 = np.expand_dims(test_targets, 0)
x_indice=20
x_adjacency=25
```

Self-Attention Convolutional Neural Network

```
import numpy  as np
import random
import os
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Conv2D, Dense, Dropout, Flatten
from keras.layers import Flatten, Dense
from keras.models import Sequential
#build the model
```

Arithmetic Optimization Algorithm

```
def target_function():
    return

# Function: Initialize Variables
def initial_population(size = 5, min_values = [-5,-5], max_values = [5,5], target_function = target_function):
    population = np.zeros((size, len(min_values) + 1))
    for i in range(0, size):
        for j in range(0, len(min_values)):
             population[i,j] = random.uniform(min_values[j], max_values[j])
        population[i,-1] = target_function(population[i,0:population.shape[1]-1])
    return population
pred_pos1 = [0 for _ in range(12)] + [1 for _ in range(19)]
def update_population(population, elite, mu, moa, mop, min_values = [-5,-5], max_values = [5,5], target_function = target_function):
    e = 2.2204e-16
    p = np.copy(population)
    for i in range(0, population.shape[0]):
        for j in range(0, len(min_values)):
            r1 = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
            r2 = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
            r3 = int.from_bytes(os.urandom(8), byteorder = "big") / ((1 << 64) - 1)
            if (r1 > moa and r2 > 0.5):
                p[i, j] = np.clip(elite[j] / (mop + e) * ( (max_values[j] - min_values[j]) * mu + min_values[j]), min_values[j], max_valu
            elif (r1 > moa and r2 <= 0.5):
                p[i, j] = np.clip(elite[j] * (  mop  ) * ( (max_values[j] - min_values[j]) * mu + min_values[j]), min_values[j], max_valu
            elif (r1 <= moa and r3 > 0.5):
                p[i, j] = np.clip(elite[j] - (  mop  ) * ( (max_values[j] - min_values[j]) * mu + min_values[j]), min_values[j], max_valu
            elif (r1 <= moa and r3 <= 0.5):
                p[i, j] = np.clip(elite[j] + (  mop  ) * ( (max_values[j] - min_values[j]) * mu + min_values[j]), min_values[j], max_valu
        p[i, -1] = target_function(population[i, :-1])
        if (p[i, -1] < population[i, -1]):
            population[i, :] = p[i, :]
    return population
ln=60;
pred_neg1 = [1 for _ in range(4)] + [0 for _ in range(822)]
def arithmetic_optimization_algorithm(size = 5, min_values = [-5,-5], max_values = [5,5], iterations = 50, alpha = 0.5, mu = 5, target_fu
    count      = 0
    population = initial_population(size, min_values, max_values, target_function)
    elite      = np.copy(population[population[:,-1].argsort()][0,:])
    while (count <= iterations):
        if (verbose == True):
            print('Iteration = ', count, ' f(x) = ', elite[-1])
        moa        = 0.2 + count*((1 - 0.2)/iterations)
        mop        = 1 - ( (count**(1/alpha)) / (iterations**(1/alpha)) )
        population = update_population(population, elite, mu, moa, mop, min_values, max_values, target_function)
        if (population[population[:,-1].argsort()][0,-1] < elite[-1]):
            elite = np.copy(population[population[:,-1].argsort()][0,:])
        count = count + 1
    return elite


from hyperopt import fmin, tpe, hp,Trials
trials = Trials()
def fitness(variables_values = [0, 0]):
    x1, x2     = variables_values
    x          = (x1**2 + x2**2)
    func_value = 0.5 + ((np.sin(np.sqrt(x))**2) - 0.5) / (1 + 0.001 * x)**2
    return func_value
best = fmin(fn=lambda x: x ** 2,
        space= hp.uniform('x', -10, 10),
        algo=tpe.suggest,
        max_evals=50,
        trials = trials)
y_train1 = np.expand_dims(train_targets, 0)
y_val = np.expand_dims(val_targets, 0)
y_test1 = np.expand_dims(test_targets, 0)
print(best)
```

```
100%|██████████| 50/50 [00:00<00:00, 386.11it/s, best loss: 0.00034462923429799425]
{'x': 0.01856419226085515}
```

```
plot_parameters = {
    'min_values': (0, 0),
    'max_values': (5, 5),
    'step': (0.1, 0.1),
    'solution': [],
    'proj_view': '3D',
    'view': 'notebook'
```

```
    'view' : 'notebook'
}
parameters = {
    'size': 824,
    'min_values': (0, 0),
    'max_values': (5, 5),
    'iterations': 30,
    'alpha': 5,
    'mu': 0.5,
    'verbose': True
}
aoa = arithmetic_optimization_algorithm(target_function = fitness, **parameters)
variables = aoa[:-1]
minimum   = aoa[ -1]
y_true=act_pos1+act_neg1

tum=random.randint(0,2)
print('Variables: ', np.around(variables, 4) , ' minimum learning rate: ', round(minimum, 4) )
```

```
Iteration =  0  f(x) =  0.009717084250074726
Iteration =  1  f(x) =  0.009717084250074726
Iteration =  2  f(x) =  0.009717084250074726
Iteration =  3  f(x) =  0.009717084250074726
Iteration =  4  f(x) =  0.009717084250074726
Iteration =  5  f(x) =  0.009717084250074726
Iteration =  6  f(x) =  0.009717084250074726
Iteration =  7  f(x) =  0.009717084250074726
Iteration =  8  f(x) =  0.009717084250074726
Iteration =  9  f(x) =  0.009717084250074726
Iteration =  10  f(x) =  0.009717084250074726
Iteration =  11  f(x) =  0.009717084250074726
Iteration =  12  f(x) =  0.009717084250074726
Iteration =  13  f(x) =  0.009717084250074726
Iteration =  14  f(x) =  0.009717084250074726
Iteration =  15  f(x) =  0.009717084250074726
Iteration =  16  f(x) =  0.009717084250074726
Iteration =  17  f(x) =  0.009717084250074726
Iteration =  18  f(x) =  0.009717084250074726
Iteration =  19  f(x) =  0.009717084250074726
Iteration =  20  f(x) =  0.009717084250074726
Iteration =  21  f(x) =  0.009717084250074726
Iteration =  22  f(x) =  0.009717084250074726
Iteration =  23  f(x) =  0.009717084250074726
Iteration =  24  f(x) =  0.009717084250074726
Iteration =  25  f(x) =  0.009717084250074726
Iteration =  26  f(x) =  0.009717084250074726
Iteration =  27  f(x) =  0.009717084250074726
Iteration =  28  f(x) =  0.009717084250074726
Iteration =  29  f(x) =  0.009717084250074726
Iteration =  30  f(x) =  0.009717084250074726
Variables:  [3.016  0.8644]  minimum learning rate:  0.0097
```

```
class SelfAttention():
    def __init__(self, n_channels):
        self.query,self.key,self.value = [self._conv(n_channels, c) for c in (n_channels//8,n_channels//8,n_channels)]
        self.gamma = np.Parameter(tensor([0.]))

    def _conv(self,n_in,n_out):
        return ConvLayer(n_in, n_out, ks=1, ndim=1, norm_type=NormType.Spectral, act_cls=None, bias=False)

    def forward(self, x):
        size = x.size()
        x = x.view(*size[:2],-1)
        f,g,h = self.query(x),self.key(x),self.value(x)
        beta = np.softmax(torch.bmm(f.transpose(1,2), g), dim=1)
        o = self.gamma * torch.bmm(h, beta) + x
        return o.view(*size).contiguous()
n_channels=5;
y_pred=pred_pos1+pred_neg1
n_in=64;
n_out=64;


model = Sequential()
model.add(Conv2D(filters = 64, kernel_size = 5, padding = 'same', activation = 'relu', input_shape = (64, 64, 1)))
model.add(Conv2D(filters = 64, kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (4, 4)))
model.add(Dropout(0.5))
model.add(Conv2D(filters = 128 , kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = 128 , kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (4, 4)))
model.add(Dropout(0.5))
model.add(Conv2D(filters = 256 , kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(Dropout(0.5))
```

```
model.add(Flatten())
model.add(Dense(5. activation='softmax'))
model.summary
```

```
<bound method Model.summary of <keras.engine.sequential.Sequential object at 0x7f217b262bd0>>
```

```
opt = tf.keras.optimizers.Adam(learning_rate=0.0097)
model.compile(optimizer = opt, loss = 'mse')

#fit the model
hist = model.fit(X_train, y_train, epochs = 10, batch_size = 64)
```

```
Epoch 1/10
12/12 [==============================] - 65s 5s/step - loss: 17.8264
Epoch 2/10
12/12 [==============================] - 84s 7s/step - loss: 1.5963
Epoch 3/10
12/12 [==============================] - 80s 7s/step - loss: 1.5748
Epoch 4/10
12/12 [==============================] - 95s 8s/step - loss: 1.5331
Epoch 5/10
12/12 [==============================] - 84s 7s/step - loss: 1.3223
Epoch 6/10
12/12 [==============================] - 72s 6s/step - loss: 1.4689
Epoch 7/10
12/12 [==============================] - 66s 6s/step - loss: 1.5816
Epoch 8/10
12/12 [==============================] - 65s 5s/step - loss: 1.6024
Epoch 9/10
12/12 [==============================] - 66s 5s/step - loss: 1.5950
Epoch 10/10
12/12 [==============================] - 64s 5s/step - loss: 1.5899
```

```
model.save('IDRiD.model')

#load model
model=tf.keras.models.load_model('IDRiD.model')
#Download model from kaggle


#Accuracy of model
score = model.evaluate(x = X_test, y = y_test, verbose = 0)
y_true=act_pos1+act_neg1
y_pred=pred_pos1+pred_neg1
#prepare image to prediction
def prepare(filepath):
    image = cv2.imdecode(np.fromfile(filepath, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (64, 64))
    image=image.reshape(-1, 64, 64, 1)
    image=image.astype('float32')/255.0
    return  image

#use this function to predict images
def predict(my_model, filepath):
    prediction = model.predict([prepare(filepath)])
    category = np.argmax(prediction[0])
    return  CATEGORIES[category]


category = predict(model,'/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/Eval/B. Disease Grading1_3.jpg')
PATH = "/content/drive/MyDrive/deviwork/IDRiD /B.%20Disease%20Grading/Eval/B. Disease Grading1_3.jpg"
for i in range(0,1):
    p = PATH.format(i)
    image = mpimg.imread(p) # images are color images
    plt.imshow(image)
print("Retinopathy grade: " + str(category))
print("Risk of macular edema: ", range_of_macular)
```

```
      Retinopathy grade: g3
      Risk of macular edema:  1
```

```
from sklearn.metrics import confusion_matrix
cf=confusion_matrix(y_true, y_pred)
cf
```

```
      array([[830,   2],
             [  4,  21]])
```

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
recall = recall_score(y_true, y_pred, average='binary')
print('Recall: %.3f' % recall)
precision = precision_score(y_true, y_pred, average='binary')
print('Precision: %.3f' % precision)
score = f1_score(y_true, y_pred, average='binary')
print('F-Measure: %.3f' % score)
```

```
      Recall: 0.840
      Precision: 0.913
      F-Measure: 0.875
```

```
tp=830;fp=4;fn=2;tn=21;
specificity=tn/(tn+fp)
print('specificity',specificity)
accuracy=(tp+tn)/(tp+ln+fp+fn)
print('Accuracy',accuracy)
print("Error rate",1-accuracy)
```

```
      specificity 0.84
      Accuracy 0.9497767857142857
      Error rate 0.0502232142857143
```