

Configuration Manual

MSc Data Analytics
Research Project

Mrunali Khokale
Student ID: x22138561

School of Computing
National College of Ireland

Supervisor: Abdul Qayum

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name	Mrunali Khokale
Student ID	X22138561
Programme	MSc in Data Analytics
Year:	2023-2024
Module:	MSc Research Project
Supervisor:	Abdul Qayum
Submission Due Date:	31/01/2024
Project Title:	Diabetic Retinopathy Detection Using Advanced Deep Learning Algorithms
Word Count:	1034
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature	Mrunali Khokale
Date	31/01/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Mrunali Khokale.
Student ID: x22138561

1 Introduction

This Configuration Manual lists all prerequisites needed to duplicate the studies and their effects on a specific setting. A glimpse of the source for Data Importing & Classifying data as per categories and after that images are augmented, and prediction algorithms are built for detection of class.

The report is organized as follows, with details relating to environment configuration provided in Section 2. Information about data gathering is detailed in Section 3. Data Classification is done in Section 4. Image Segmentation is included in Section 5. In section 6, the Image Augmentation is described. Details about models that were created and tested are provided in Section 7. How the results are calculated and shown is described in Section 8.

2 System Requirements

The specific needs for hardware and software to put the research into use are detailed in this section.

2.1 Hardware Requirements

2.1 Hardware Requirements

The code execution took place on my HP Spectre x360 laptop, featuring the hardware specifications detailed in Figure 1:

- Processor: Intel Core i7 12th Gen
- Operating System: 64-bit
- RAM: 16GB
- Storage: 256GB SSD
- Display: 14-inch

It is anticipated that the code is compatible with a configuration comprising an i3 processor, 8GB RAM, and a 256GB SSD. However, it is pertinent to mention that no specific testing has been conducted on this setup.

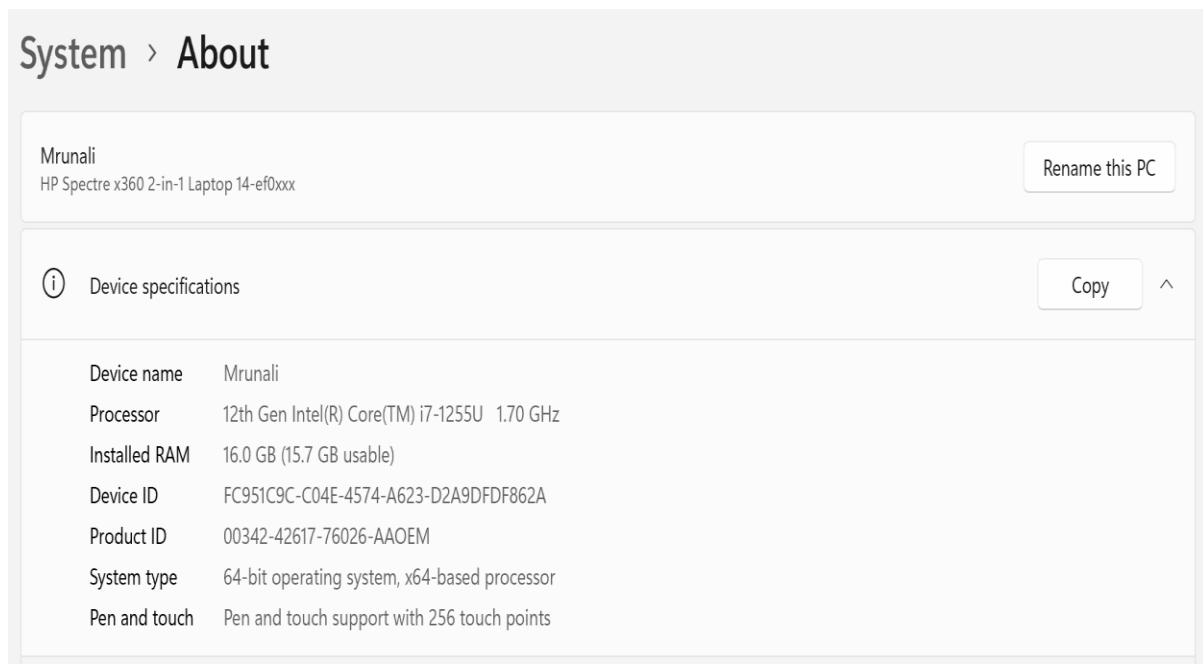


Figure 1. Hardware Requirement

2.2 Software Requirements

- ☐ Anaconda 3 (Version 4.8.0)
- ☐ Jupyter Notebook (Version 6.0.3)
- ☐ Python (Version 3.7.6)

2.3 Code Execution

The code can be run in jupyter notebook. The Jupyter Notebook comes with Anaconda 3, running the Jupyter Notebook from startup. This will open Jupyter Notebook in the web browser. The web browser will show the folder structure of the system and move to the folder where the code file is located. Open the code file from the folder and to run the code, go to the Kernel menu and run all cells.

3 Data Collection

The dataset is taken from the Kaggle public repository from the link <https://www.kaggle.com/competitions/diabetic-retinopathy-detection/data>. a large set of high-resolution retina images taken under a variety of imaging conditions. A left and right field is provided for every subject. Images are labeled with a subject ID as well as either left or right (e.g. 1_left.jpeg is the left eye of the patient id 1)

4 Data Classification

Figure 2 lists every Python library necessary to complete the project.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
import warnings
import os
import shutil
warnings.filterwarnings('ignore')
import glob
import cv2
from PIL import ImageFile
import tensorflow
import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Conv2D, Dense, Flatten, Layer, Input, Dropout, BatchNormalization
from tensorflow.keras.layers import Attention, MultiHeadAttention, LayerNormalization, Add, Multiply, MaxPooling2D
from keras.applications.inception_v3 import InceptionV3
from keras.applications.resnet_v2 import ResNet50V2
from keras.applications.efficientnet_v2 import EfficientNetV2L
```

Figure 2: Necessary Python libraries

Figure 3 represents the block of code to import the training videos and check for the data.

```
#Importing Data Set CSV file
trainLabels=pd.read_csv(r'trainLabels.csv')
```

```
#Dataset Information
trainLabels.head(10)
```

	image	level
0	10_left	0
1	10_right	0
2	13_left	0
3	13_right	0
4	15_left	1
5	15_right	2
6	16_left	4
7	16_right	4
8	17_left	0
9	17_right	1

Figure 3: Importing training videos and Checking Data Information

As seen in Figure 4, appending '.jpeg' in the image column to complete the filename.

```
trainLabels['image']= trainLabels['image']+'.jpeg'
trainLabels
```

	image	level
0	10_left.jpeg	0
1	10_right.jpeg	0
2	13_left.jpeg	0
3	13_right.jpeg	0
4	15_left.jpeg	1
...	...	...
35121	44347_right.jpeg	0
35122	44348_left.jpeg	0
35123	44348_right.jpeg	0
35124	44349_left.jpeg	0
35125	44349_right.jpeg	1

35126 rows × 2 columns

Figure 4: Image file data

Figure 5, the code generates a subset of data based on classes of images and creates a Data folder to classify data based on each category in the folder.

```
#Seperating 0 - No DR 1 - Mild 2 - Moderate 3 - Severe 4 - Proliferative DR into different dataset

NoDR = trainLabels.loc[(trainLabels['level']==0),:]
Mild = trainLabels.loc[(trainLabels['level']==1),:]
Moderate = trainLabels.loc[(trainLabels['level']==2),:]
Severe = trainLabels.loc[(trainLabels['level']==3),:]
Proliferative = trainLabels.loc[(trainLabels['level']==4),:]

IMAGE_PATH= './train/train/' # source file path
Destination_path='Data' # destination file path
if os.path.exists(Destination_path):
    pass
else:
    os.makedirs(Destination_path)

Image_Data=os.listdir(IMAGE_PATH)
```

Figure 5: Data Categorization

Figure 6, illustrates the code to classify images into folders based on their category.

```
def make_direct(Disease,image):
    global Diseases_path
    if image in Image_Data:
        if os.path.exists(os.path.join(Destination_path+Disease)):
            pass
        else:
            os.makedirs(Destination_path+Disease)
            print(image)
    return shutil.copyfile(os.path.join(IMAGE_PATH,image),os.path.join((Destination_path+Disease),image))

#Created Directory for Moderate Diabetic Retinopathy
moderate = [Moderate['image'].apply(lambda x: make_direct(r'\Moderate',x))]
moderate

[5      Data\Moderate\15_right.jpeg
 23                                     None
 30                                     None
 42                                     None
 46                                     None
...
35005                                     None
35018                                     None
35019                                     None
35034                                     None
35063                                     None
Name: image, Length: 5292, dtype: object]

#Created Directory for Severe Diabetic Retinopathy
severe = [Severe['image'].apply(lambda x: make_direct(r'\Severe',x))]
severe

[90                                     None
 91                                     None
134      Data\Severe\163_left.jpeg
135      Data\Severe\163_right.jpeg
283                                     None
...
34915                                     None
34961                                     None
34994                                     None
34996                                     None
34997                                     None
Name: image, Length: 873, dtype: object]

#Created Directory for Mild Diabetic Retinopathy
mild = [Mild['image'].apply(lambda x: make_direct(r'\Mild',x))]
mild

[4      Data\Mild\15_left.jpeg
 9      Data\Mild\17_right.jpeg
22                                     None
28                                     None
102     Data\Mild\114_left.jpeg
...
35098                                     None
35099                                     None
35105                                     None
35112                                     None
35125                                     None
Name: image, Length: 2443, dtype: object]

#Created Directory for Proliferative Diabetic Retinopathy
proliferative = [Proliferative['image'].apply(lambda x: make_direct(r'\Proliferative',x))]
proliferative

[6      Data\Proliferative\16_left.jpeg
 7      Data\Proliferative\16_right.jpeg
184     Data\Proliferative\217_left.jpeg
185     Data\Proliferative\217_right.jpeg
252                                     None
...
34914                                     None
35046                                     None
35047                                     None
35048                                     None
35049                                     None
Name: image, Length: 708, dtype: object]
```

Figure 6: Data Categorization

Figure 7 illustrates the code to update the data path and set image size and categories.

```
# PATH Initilization
DATA_PATH=Destination_path

# Initilizing parameters related to images
IMAGE_WIDTH= 256
IMAGE_HEIGHT= 256

path_noDR = './Data/NoDR/'
path_mild = './Data/Mild/'
path_moderate = './Data/Moderate/'
path_severe = './Data/Severe/'
path_proliferative = './Data/Proliferative/'

categories = ['NoDR', 'Mild', 'Moderate', 'Severe', 'Proliferative']
print(categories)

['NoDR', 'Mild', 'Moderate', 'Severe', 'Proliferative']
```

Figure 7: Data Path

Figure 8 and Figure 9, illustrate the check count of images in each category and generate a bar plot of the value counts.



Figure 8: Data Classes

```
categories = ['No DR', 'Mild', 'Moderate', 'Severe', 'Proliferative']
values = [count_nodr, count_mild, count_moderate, count_sevre, count_proliferative]

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(values, labels=categories, autopct='%1.1f%%', startangle=140, colors=['#66b3ff', '#66b3ff', '#66b3ff', '#66b3ff', '#66b3ff'])
plt.title('Distribution of Classes in the Dataset (Pie Chart)')
plt.show()
```

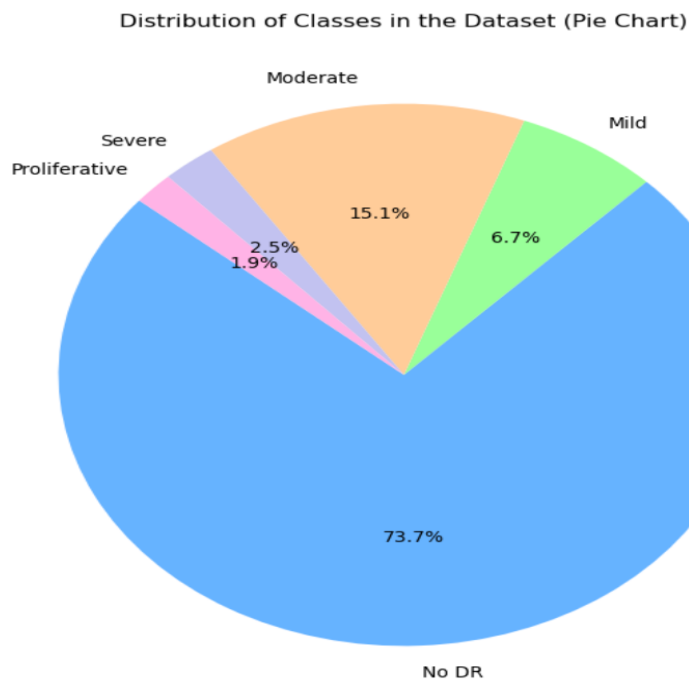


Figure 9: Data Classes Pie chart

Figure 10 illustrates the generated list of images for each class.

```
nodr = glob.glob(path_nodr + "*.jpeg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d images.\nFirst 5 filenames:" % len(nodr))
print ('\n'.join(nodr[:5]))
```

```
Total of 12387 images.
First 5 filenames:
./Data/NoDR\10003_left.jpeg
./Data/NoDR\10003_right.jpeg
./Data/NoDR\10007_left.jpeg
./Data/NoDR\10007_right.jpeg
./Data/NoDR\10009_left.jpeg
```

```
mild = glob.glob(path_mild + "*.jpeg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d images.\nFirst 5 filenames:" % len(mild))
print ('\n'.join(mild[:5]))
```

```
Total of 1130 images.
First 5 filenames:
./Data/Mild\10030_left.jpeg
./Data/Mild\10030_right.jpeg
./Data/Mild\10085_left.jpeg
./Data/Mild\10085_right.jpeg
./Data/Mild\10150_right.jpeg
```

```
moderate = glob.glob(path_moderate + "*.jpeg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d images.\nFirst 5 filenames:" % len(moderate))
print ('\n'.join(moderate[:5]))
```

```
Total of 2537 images.
First 5 filenames:
./Data/Moderate\1002_right.jpeg
./Data/Moderate\10043_left.jpeg
./Data/Moderate\10043_right.jpeg
./Data/Moderate\10109_left.jpeg
./Data/Moderate\10109_right.jpeg
```

```
severe = glob.glob(path_severe + "*.jpeg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d images.\nFirst 5 filenames:" % len(severe))
print ('\n'.join(severe[:5]))
```

```
Total of 424 images.
First 5 filenames:
./Data/Severe\1002_left.jpeg
./Data/Severe\10047_left.jpeg
./Data/Severe\1008_left.jpeg
./Data/Severe\1008_right.jpeg
./Data/Severe\10125_left.jpeg
```

```
proliferative = glob.glob(path_proliferative + "*.jpeg")
# Print out the first 5 file names to verify we're in the right folder.
print ("Total of %d images.\nFirst 5 filenames:" % len(proliferative))
print ('\n'.join(proliferative[:5]))
```

```
Total of 324 images.
First 5 filenames:
./Data/Proliferative\10017_left.jpeg
./Data/Proliferative\10017_right.jpeg
./Data/Proliferative\10047_right.jpeg
./Data/Proliferative\10193_right.jpeg
./Data/Proliferative\10312_left.jpeg
```

Figure 10: Images in each class

5 Image Segmentation

Figure 11 illustrates the read the image and shows the image shape and plot image.

```
def plot_image(img, cmap='gray'):
    fig = plt.figure(figsize=(8,8))
    axes = fig.add_subplot(111)
    axes.imshow(img, cmap=cmap)

img1 = cv2.imread(nodr[1])
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
plot_image(img1)
width, height, dimension = img1.shape
print(f'Width RGB = {width}')
print(f'Height RGB = {height}')
print(f'Dimension RGB = {dimension}')

Width RGB = 3168
Height RGB = 4752
Dimension RGB = 3
```



Figure 11: Read Image

Figures 12 show the code used to read the image in gray scale.

```
img1_gray = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
plot_image(img1_gray)
width, height = img1_gray.shape
print(f'Width Grayscale = {width}')
print(f'Height Grayscale = {height}')
print(f'Image Shape Grayscale {img1_gray.shape}')

Width Grayscale = 3168
Height Grayscale = 4752
Image Shape Grayscale (3168, 4752)
```

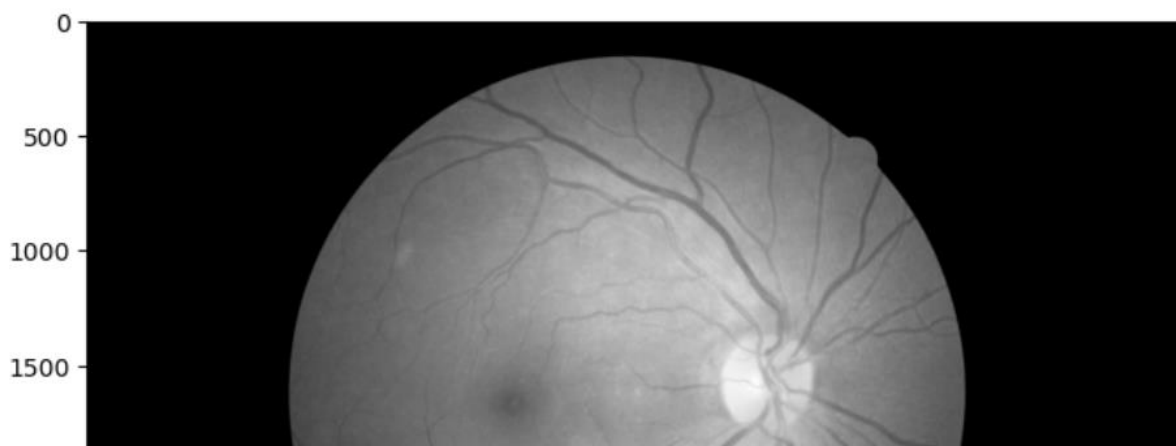
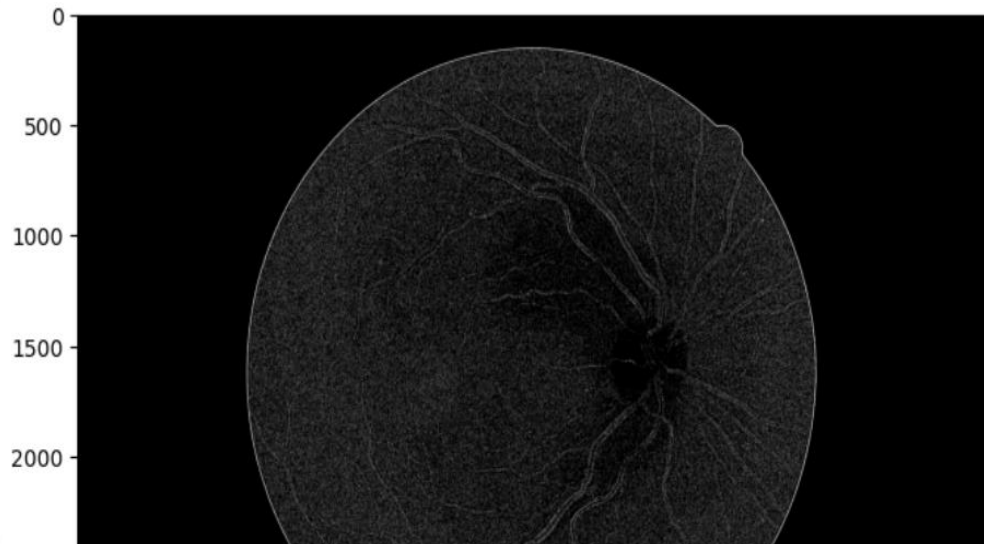


Figure 12: Read GrayScale Image

Figure 13 illustrates the code to generate adaptive thresholds of the images and display the contours.

```
img1_gray = cv2.adaptiveThreshold(img1_gray,5,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,11,3)
plot_image(img1_gray)
```



```
contours = cv2.findContours(img1_gray, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = contours[0] if len(contours) == 2 else contours[1]
contours = sorted(contours, key=cv2.contourArea, reverse=True)
for c in contours:
    x,y,w,h = cv2.boundingRect(c)
    img1_ROI = img1[y:y+h, x:x+w]
    break
plot_image(img1_ROI)
width, height, dimension = img1_ROI.shape
print(f'Width = {width}')
print(f'Height = {height}')
print(f'Dimension = {dimension}')
```

```
Width = 2939
Height = 2958
Dimension = 3
```

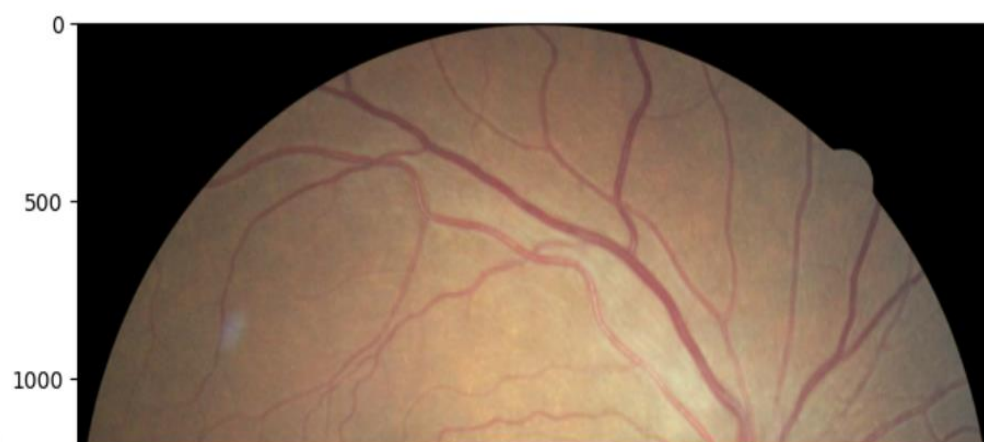


Figure 13: Adaptive threshold and contouring

6 Image Augmentation

Figure 14 illustrates the code to use ImageDataGenerator to generate augmented images for the deep learning models.

```
del noDR, mild, moderate, severe, proliferative, trainLabels

img_size = 100

gen = ImageDataGenerator(rescale=1./255)

train = gen.flow_from_directory(directory=DATA_PATH, target_size=(img_size,img_size), class_mode='sparse')

Found 16802 images belonging to 5 classes.

# This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 4, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()

augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

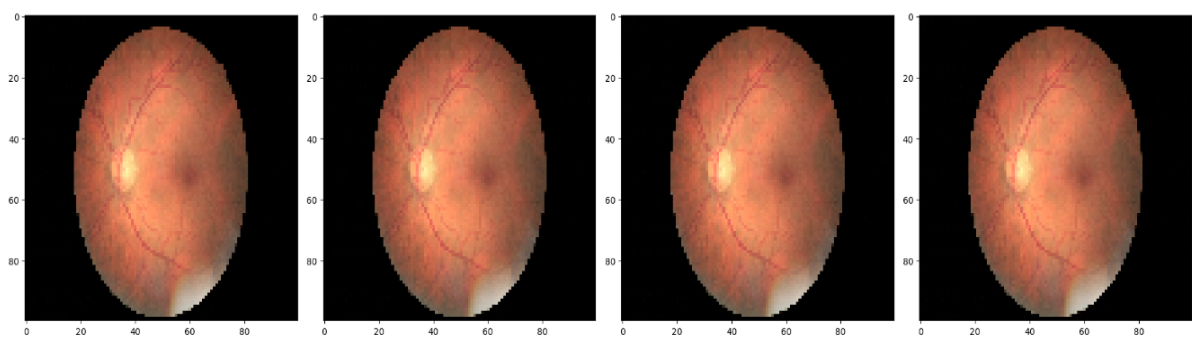


Figure 14: Image Data Generator

Figures 15 show the code to create data with different height and width shifts to check for appropriate size.

```
gen = ImageDataGenerator(rescale=1./255, width_shift_range=0.2, height_shift_range=0.3)
train = gen.flow_from_directory(directory=DATA_PATH, target_size=(img_size,img_size), class_mode='sparse')
augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)

Found 16802 images belonging to 5 classes.
```

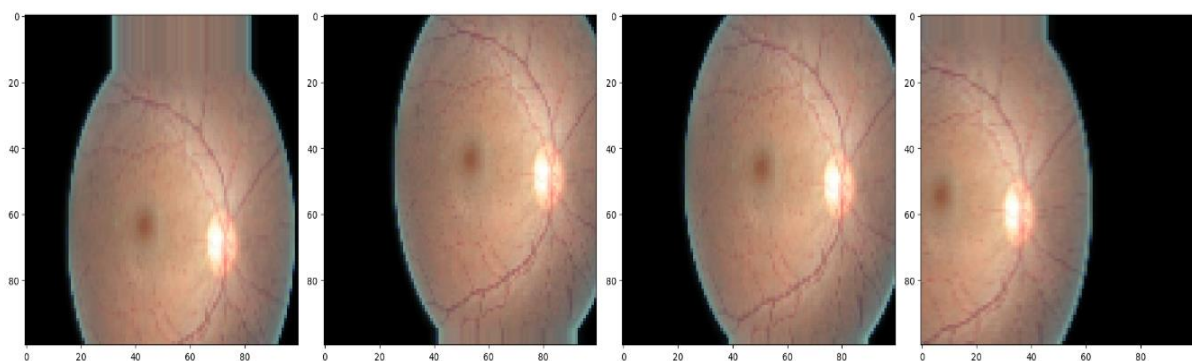


Figure 15: Image Data Generator

Figure 16 shows the code to create data by rescaling the images.

```
gen = ImageDataGenerator(rescale=1./255)
train = gen.flow_from_directory(directory=DATA_PATH, target_size=(img_size,img_size))
augmented_images = [train[0][0][0] for i in range(4)]
plotImages(augmented_images)
```

Found 16802 images belonging to 5 classes.

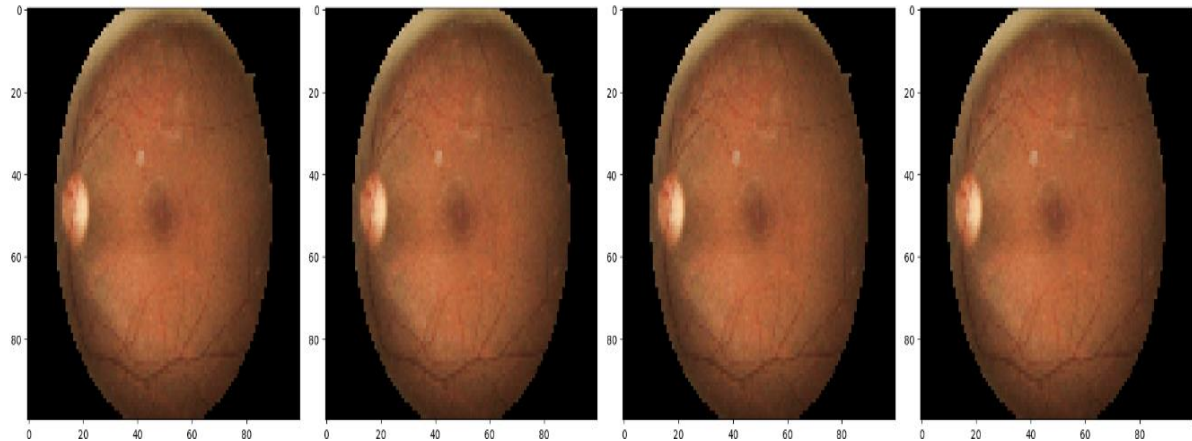


Figure 16: Image Data Generator

Figure 17 illustrates the code to data and validation split to generate train and test data.

```
datagen= ImageDataGenerator(rescale=1./255, validation_split=0.1)
train = datagen.flow_from_directory(DATA_PATH, target_size=(img_size, img_size), subset='training', shuffle=True,
    classes=categories)

test = datagen.flow_from_directory(DATA_PATH, target_size=(img_size, img_size), subset='validation', shuffle=True,
    classes=categories)
```

Found 15124 images belonging to 5 classes.

Found 1678 images belonging to 5 classes.

```
input_shape = (img_size, img_size, 3)
input_shape
```

(100, 100, 3)

Figure 17: Image Data Generator

7 Machine Learning Models

7.1 Transformer CNN

```
# Encoder #####
input = Input(shape=input_shape)
dense_layer = Dense(3, activation = "relu")(input)
attention_layer = Attention(score_mode="dot")([input, dense_layer])
x = Add()([input, attention_layer]) # residual connection
x = LayerNormalization(epsilon=1e-6)(input)
## Feed Forward
dense = Dense(128, activation='relu')(x)
dense = Dense(3)(dense)
x = Add()([x, dense]) # residual connection
encoder = LayerNormalization(epsilon=1e-6)(x)
```

```
# Decoder #####
target = Input(shape=input_shape)
## positional encoding
scaling_factor = keras.backend.constant(np.sqrt(3), shape = input_shape)
x = Multiply()([target, scaling_factor])
dense_layer = Dense(3, activation = "relu")(target)
attention_layer = Attention(score_mode="dot")([target, dense_layer])
x = Add()([x, attention_layer]) # residual connection
x = LayerNormalization(epsilon=1e-1)(input)
## Feed Forward
dense = Dense(128, activation='relu')(x)
dense = Dense(3)(dense)
x = Add()([x, dense]) # residual connection
decoder = LayerNormalization(epsilon=1e-6)(x)
```

```
flatten = Flatten()(decoder)
x = Dense(5, activation='relu')(flatten)
model = Model(inputs=input, outputs=x)
model.summary()
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
```

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 100, 100, 3)]	0	[]
layer_normalization_6 (LayerNormalization)	(None, 100, 100, 3)	6	['input_3[0][0]']
dense_11 (Dense)	(None, 100, 100, 128)	512	['layer_normalization_6[0][0]']
dense_12 (Dense)	(None, 100, 100, 3)	387	['dense_11[0][0]']
add_7 (Add)	(None, 100, 100, 3)	0	['layer_normalization_6[0][0]', 'dense_12[0][0]']
layer_normalization_7 (LayerNormalization)	(None, 100, 100, 3)	6	['add_7[0][0]']

```
model.fit(train)
```

473/473 [=====] - 2389s 5s/step - loss: 2.7787 - accuracy: 0.7073

<keras.src.callbacks.History at 0x1e18df84910>

Figure 18: Implementation of Transformer CNN

7.2 ResNet 50

```
model = ResNet50V2(include_top=False, weights='imagenet', input_shape=input_shape)
model.summary()
```

Model: "resnet50v2"

Layer (type)	Output Shape	Param #	Connected to
input_5 (InputLayer)	[(None, 100, 100, 3)]	0	[]
conv1_pad (ZeroPadding2D)	(None, 106, 106, 3)	0	['input_5[0][0]']
conv1_conv (Conv2D)	(None, 50, 50, 64)	9472	['conv1_pad[0][0]']
pool1_pad (ZeroPadding2D)	(None, 52, 52, 64)	0	['conv1_conv[0][0]']
pool1_pool (MaxPooling2D)	(None, 25, 25, 64)	0	['pool1_pad[0][0]']
conv2_block1_preact_bn (Batch Normalization)	(None, 25, 25, 64)	256	['pool1_pool[0][0]']
conv2_block1_preact_relu (Activation)	(None, 25, 25, 64)	0	['conv2_block1_preact_bn[0][0]']

```
x = model.output
x = Flatten()(x)
x = Dense(512, activation='tanh')(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='tanh')(x)
x = Dropout(0.01)(x)
x = Dense(64, activation='tanh')(x)
x = Dropout(0.05)(x)
output_layer = Dense(1, activation='sigmoid')(x)

model = Model(inputs=model.input, outputs=output_layer)
```

```
model.compile(optimizer = 'adam', loss= 'binary_crossentropy', metrics = ['accuracy'])
model.summary()
```

Model: "model_2"

Layer (type)	Output Shape	Param #	Connected to
input_5 (InputLayer)	[(None, 100, 100, 3)]	0	[]
conv1_pad (ZeroPadding2D)	(None, 106, 106, 3)	0	['input_5[0][0]']
conv1_conv (Conv2D)	(None, 50, 50, 64)	9472	['conv1_pad[0][0]']
pool1_pad (ZeroPadding2D)	(None, 52, 52, 64)	0	['conv1_conv[0][0]']
pool1_pool (MaxPooling2D)	(None, 25, 25, 64)	0	['pool1_pad[0][0]']
conv2_block1_preact_bn (Batch Normalization)	(None, 25, 25, 64)	256	['pool1_pool[0][0]']
conv2_block1_preact_relu (Activation)	(None, 25, 25, 64)	0	['conv2_block1_preact_bn[0][0]']

```
model.fit(train, batch_size=500)
```

473/473 [=====] - 7814s 15s/step - loss: 0.5070 - accuracy: 0.7996

<keras.src.callbacks.History at 0x1e1815f0be0>

Figure 19: Implementation of ResNet50

7.3 InceptionNet

```
base_model = InceptionV3(weights='imagenet', include_top=False)
base_model.summary
```

<bound method Model.summary of <keras.src.engine.functional.Functional object at 0x000001E185E74D60>>

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.15)(x)
x = Dense(128, activation='sigmoid')(x)
x = Dropout(0.05)(x)
x = Dense(8, activation='sigmoid')(x)
predictions = Dense(5, activation='sigmoid')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

```
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

Model: "model_3"

Layer (type)	Output Shape	Param #	Connected to
=====			
input_6 (InputLayer)	[(None, None, None, 3)]	0	[]
conv2d (Conv2D)	(None, None, None, 32)	864	['input_6[0][0]']
batch_normalization (Batch Normalization)	(None, None, None, 32)	96	['conv2d[0][0]']
activation (Activation)	(None, None, None, 32)	0	['batch_normalization[0][0]']
conv2d_1 (Conv2D)	(None, None, None, 32)	9216	['activation[0][0]']
batch_normalization_1 (Batch Normalization)	(None, None, None, 32)	96	['conv2d_1[0][0]']
activation_1 (Activation)	(None, None, None, 32)	0	['batch_normalization_1[0][0]']

```
model.fit(train, batch_size=5000)
```

473/473 [=====] - 3399s 7s/step - loss: 0.9909 - accuracy: 0.7331

<keras.src.callbacks.History at 0x1e180f66f40>

Figure 20: Implementation of InceptionNet V3

7.4 EfficientNet

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

```
base_model = EfficientNetV2L(weights='imagenet', include_top=False)
base_model.summary
```

<bound method Model.summary of <keras.src.engine.functional.Functional object at 0x000001E200EFA850>>

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.05)(x)
x = Dense(8, activation='sigmoid')(x)
predictions = Dense(5, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

```
model.compile(optimizer = 'adam', loss= 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

Model: "model_4"

Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	[(None, None, None, 3)]	0	[]
rescaling (Rescaling)	(None, None, None, 3)	0	['input_7[0][0]']
stem_conv (Conv2D)	(None, None, None, 32)	864	['rescaling[0][0]']
stem_bn (BatchNormalization)	(None, None, None, 32)	128	['stem_conv[0][0]']
stem_activation (Activation)	(None, None, None, 32)	0	['stem_bn[0][0]']
block1a_project_conv (Conv2D)	(None, None, None, 32)	9216	['stem_activation[0][0]']
block1a_project_bn (BatchNormalization)	(None, None, None, 32)	128	['block1a_project_conv[0][0]']

```
model.fit(train, batch_size=500)
```

473/473 [=====] - 7400s 15s/step - loss: 0.8902 - accuracy: 0.7359

<keras.src.callbacks.History at 0x1e200ef6160>

Figure 21: Implementation of EfficientNet

References

<https://www.kaggle.com/competitions/diabetic-retinopathy-detection/data>

OpenCV: OpenCV modules

Keras Applications

<https://www.picsellia.com/post/are-transformers-replacing-cnns-in-object-detection>