National College of Ireland

Edge Alterations as Predictive Biomarkers in Diabetic Retinopathy: A Deep Learning Approach Configuration Manual

MSc Research Project
Data Analytics

Purvesh Lalit Bhave
Student ID: X21220182@student.ncirl.ie

School of Computing
National College of Ireland

Supervisor: Aaloka Anant

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Purvesh Lalit Bhave
Student ID:	X21220182@student.ncirl.ie
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Aaloka Anant
Submission Due Date:	14/12/2023
Project Title:	Edge Alterations as Predictive Biomarkers in Diabetic Retinopathy: A Deep Learning Approach Configuration Manual
Word Count:	1500
Page Count:	8

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:	
Date:	13th December 2023

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):	

Edge Alterations as Predictive Biomarkers in Diabetic Retinopathy: A Deep Learning Approach Configuration Manual

Purvesh Lalit Bhave
X21220182@student.ncirl.ie

1 Introduction

This document outlines the detailed instructions about how the research was conducted, and on what system under and what environment. This manual also discusses the system configuration on which the research was carried out. The manual contains the installation of the necessary library or packages and also the minimum configuration for implementing this project.

2 File Details

Python programming language is used while executing this project. The Jupyter Notebook was used as IDE to implement the desired steps in the methodology.

2.1 Submission

The submitted zip file(x21220182Thesis) contains the code artifact of the implementation, used datasets, a report that documents what is done in the project, and a video presentation of the project.

3 System Specification

Based on the image you provided, here is a system specification description:
- Device Name: ASUS_1898
- Processor: 12th Gen Intel(R) Core(TM) i7-12700H, with a base speed of 2.30 GHz
- Installed RAM: 16.0 GB (15.6 GB usable)
- Device ID: 9322E6B9-DA90-40C5-AC78-7E0A9275E472
- Product ID: 00342-42610-97830-AAOEM
- System Type: 64-bit operating system, x64-based processor
- Pen and Touch: No pen or touch input is available for this display

Windows Specifications:
- Edition: Windows 11 Home Single Language

- Version: 22H2
- Installed on: 6/19/2023
- OS Build: 22621.2715
- Experience: Windows Feature Experience Pack 1000.22677.1000.0

Manufacturer: ASUSTeK COMPUTER INC.

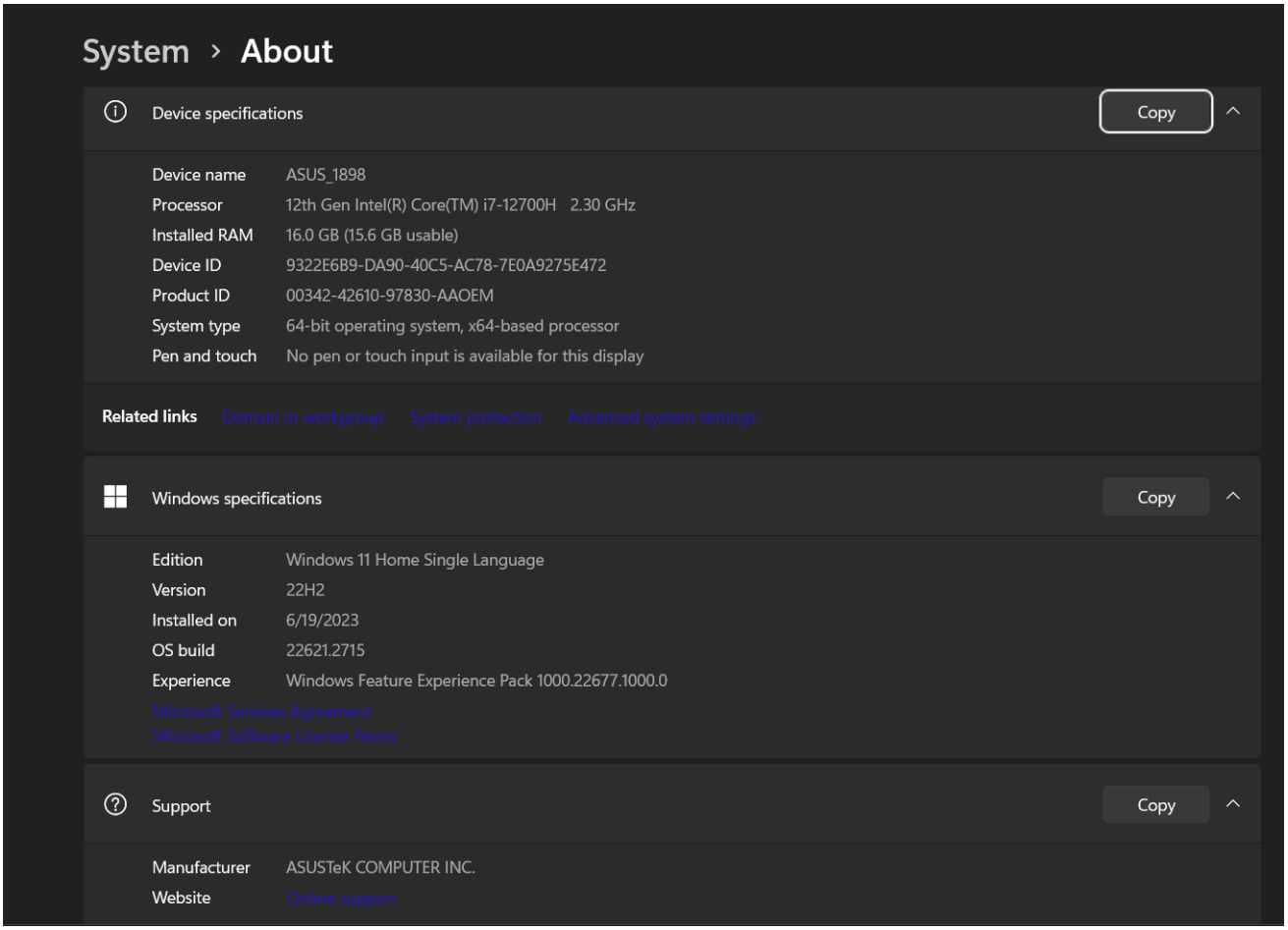


Figure 1: System Specifications

4 Softwares Used

- **Microsoft Excel:** Used for initial data exploration.
- **Canva:** This was used to make the flowcharts.
- **Python Programming:** Used for implementing the project.
- **Jupyter Notebook:** This was used as an integrated development environment for carrying out the experiments.

5 Downloading the Requirements

Python programming was first installed, and the latest version of Python is advised to download in this case version 3.12.0 was installed. As an integrated development environment, Jupyter Notebook was downloaded from the Anaconda Navigator. This IDE is very common and user-friendly for programmers to use, this IDE can be downloaded from the Anaconda Navigator's Python bundle. In the figure below the dashboard of Anaconda Navigator is shown the dashboard may vary based on the version of the Anaconda Navigator. There are multiple pre-installed IDEs or packages that can be used as per the requirements.

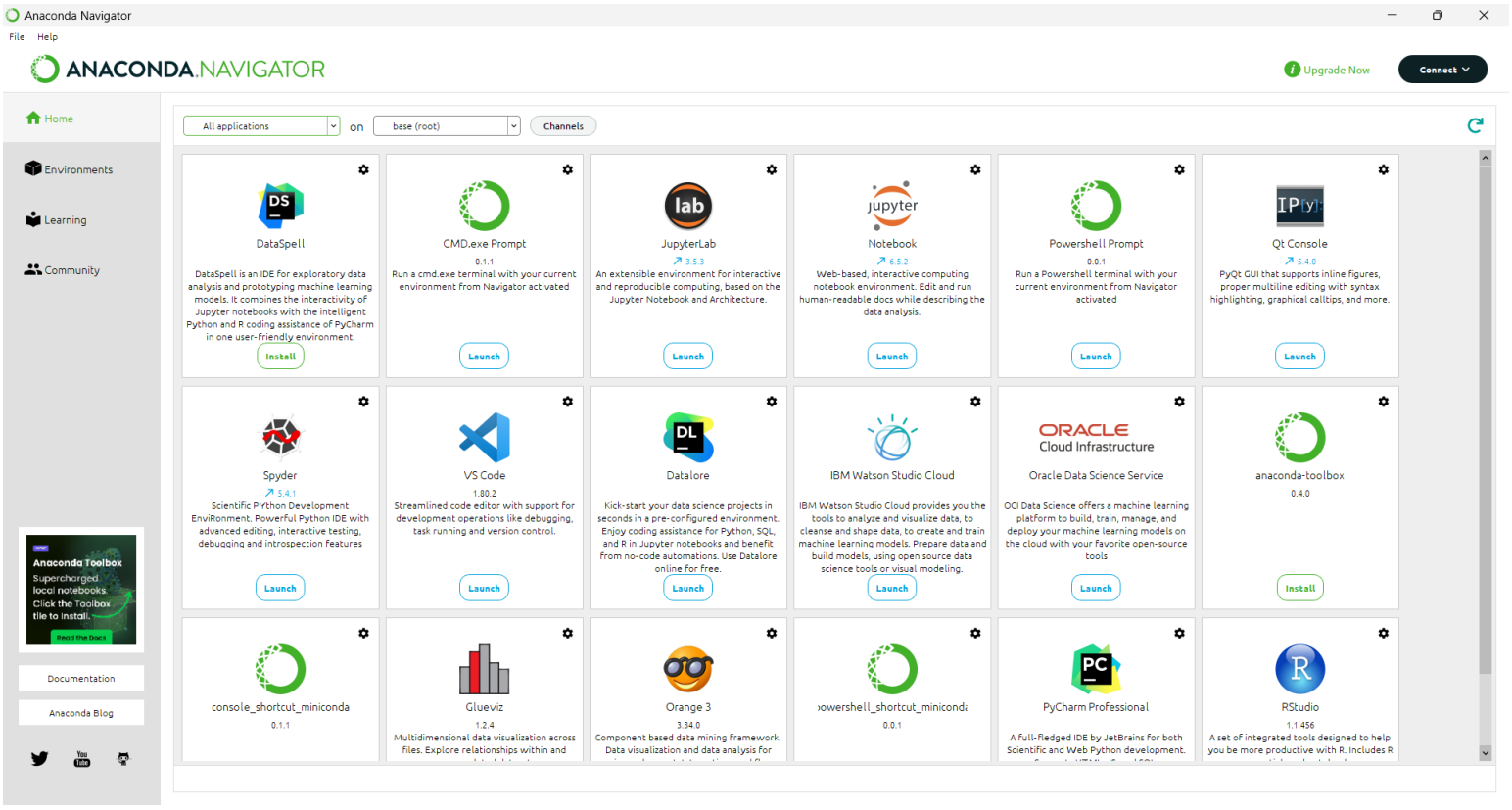


Figure 2: Anaconda Navigator Dasboard

6 Project Initiation

Once the installation of the software was done and the desired requirements were met, the implementation of the project was started. Before the initiation of the project, some basic libraries and packages were installed which eases the process of programming. Many more such libraries and packages were installed through out the implementation. The attested screen shot below shows the basic packages that were installed before the implementation.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from PIL import Image
from skimage.io import imread
from tensorflow.keras.utils import to_categorical
import os
from sklearn.model_selection import train_test_split
from glob import glob
import warnings
warnings.filterwarnings('ignore', message='TensorFlow Addons.*')
%matplotlib inline
```

Figure 3: System Specifications

All of the above libraries can be installed if they don't exist on your environment. The pip install command with the name of the package can be used to download and install the desired packages.

6.1 Importing data

```
# Training set
retina_train_df = pd.read_csv(os.path.join("C:\\Users\\purve\\Desktop\\My Thesis\\exp1\\attempt 2\\truths1\\train.csv"))
image_dir = "Dataset\\Training_Set1" # Update this with the path to your images

retina_train_df.rename(columns={'id_code': 'image','diagnosis':'grade'}, inplace=True)
retina_train_df['path'] = retina_train_df['image'].apply(lambda x: os.path.join(image_dir, x + ".png"))

# Function to create the list based on the grade
def create_grade_list(grade):
    grade_list = [0.0] * 5
    grade_list[grade] = 1.0
    return grade_list

# Apply the function to the 'grade' column and create a new column
retina_train_df['level_cat'] = retina_train_df['grade'].apply(create_grade_list)

# Function to load an image
def load_image(image_path):
    try:
        with Image.open(image_path) as img:
            return img
    except IOError:
        return None

# Load images and store them in a new column
retina_train_df['scans'] = retina_train_df['path'].apply(load_image)

#Delete below line to run code on all images
# retina_train_df = retina_train_df.head(1000)
```

Figure 4: Data Importing

The above code snippet shows the importing the datasets and merging them. This was the first step after starting the project.

6.2 Pre-processing

```
IMG_SIZE = (256, 256) # slightly smaller than vgg16 normally expects
def tf_image_loader(out_size,
                    horizontal_flip = True,
                    vertical_flip = False,
                    random_brightness = True,
                    random_contrast = True,
                    random_saturation = True,
                    random_hue = True,
                    color_mode = 'rgb',
                    preproc_func = preprocess_input,
                    on_batch = False):
    def _func(X):
        with tf.name_scope('image_augmentation'):
            with tf.name_scope('input'):
                X = tf.image.decode_png(tf.io.read_file(X), channels = 3 if color_mode == 'rgb' else 0)
                X = tf.image.resize(X, out_size)
            with tf.name_scope('augmentation'):
                if horizontal_flip:
                    X = tf.image.random_flip_left_right(X)
                if vertical_flip:
                    X = tf.image.random_flip_up_down(X)
                if random_brightness:
                    X = tf.image.random_brightness(X, max_delta = 0.1)
                if random_saturation:
                    X = tf.image.random_saturation(X, lower = 0.75, upper = 1.5)
                if random_hue:
                    X = tf.image.random_hue(X, max_delta = 0.15)
                if random_contrast:
                    X = tf.image.random_contrast(X, lower = 0.75, upper = 1.5)
                return preproc_func(X)
    if on_batch:
        # we are meant to use it on a batch
        def _batch_func(X, y):
            return tf.map_fn(_func, X), y
        return _batch_func
    else:
        # we apply it to everything
        def _all_func(X, y):
            return _func(X), y
        return _all_func

def tf_augmentor(out_size,
                 intermediate_size = (640, 640),
                 intermediate_trans = 'crop',
                 horizontal_flip = True,
                 vertical_flip = False,
                 random_brightness = True,
                 random_contrast = True,
                 random_saturation = True,
                 random_hue = True,
                 color_mode = 'rgb',
                 preproc_func = preprocess_input,
                 min_crop_percent = 0.001,
                 max_crop_percent = 0.005,
                 crop_probability = 0.5,
                 rotation_range = 5):

    load_ops = tf_image_loader(out_size = intermediate_size,
                               horizontal_flip=horizontal_flip,
                               vertical_flip=vertical_flip,
                               random_brightness = random_brightness,
                               random_contrast = random_contrast,
                               random_saturation = random_saturation,
                               random_hue = random_hue,
                               color_mode = color_mode,
                               preproc_func = preproc_func,
                               on_batch=False)
    def create_rotation_matrix(angle):
        return [tf.math.cos(angle), -tf.math.sin(angle), 0.0,tf.math.sin(angle), tf.math.cos(angle), 0.0,0.0, 0.0, 1.0]

    def true_fn(X, transforms):
        composed_transforms = tfa.image.compose_transforms(*transforms)
        return tfa.image.transform(X, composed_transforms, interpolation='BILINEAR')
```

Figure 5: Preprocessing -1

```
def flow_from_dataframe(idg,
                        in_df,
                        path_col,
                        y_col,
                        shuffle = True,
                        batch_size = 32,
                        color_mode = 'rgb'):
    files_ds = tf.data.Dataset.from_tensor_slices((in_df[path_col].values,
                                                   np.stack(in_df[y_col].values,0)))
    in_len = in_df[path_col].values.shape[0]
    while True:
        if shuffle:
            files_ds = files_ds.shuffle(in_len) # shuffle the whole dataset

#         next_batch = idg(files_ds, batch_size).repeat().make_one_shot_iterator().get_next()
        next_batch = next(iter(idg(files_ds, batch_size).repeat()))

        for i in range(max(in_len//batch_size,1)):
            # NOTE: if we loop here it is 'thread-safe-ish' if we loop on the outside it is completely unsafe
#             yield K.get_session().run(next_batch)
            if isinstance(next_batch, tuple) and len(next_batch) == 2:
                # assuming the tuple contains images and labels
                images, labels = next_batch
                if isinstance(images, tf.Tensor):
                    images = images.numpy()
                if isinstance(labels, tf.Tensor):
                    labels = labels.numpy()
                yield images, labels
            else:
                yield next_batch


def get_iters_from_df(idg, in_df, path_col, y_col, shuffle=True, batch_size=32, color_mode='rgb'):
    files_ds = tf.data.Dataset.from_tensor_slices((in_df[path_col].values,
                                                   np.stack(in_df[y_col].values,0).astype(np.float32)))
    in_len = in_df[path_col].values.shape[0]
    if shuffle:
            files_ds = files_ds.shuffle(in_len) # shuffle the whole dataset
    idg_batches = idg(files_ds, batch_size)
    img_batch = idg_batches.map(lambda x,y: x).repeat()
    val_batch = idg_batches.map(lambda x,y: y).repeat()
    return iter(img_batch), iter(val_batch)
```

```
batch_size = 48
core_idg = tf_augmentor(out_size = IMG_SIZE,
                        color_mode = 'rgb',
                        vertical_flip = True,
                        crop_probability=0.0, # crop doesn't work yet
                        )
valid_idg = tf_augmentor(out_size = IMG_SIZE, color_mode = 'rgb',
                         crop_probability=0.0,
                         horizontal_flip = False,
                         vertical_flip = False,
                         random_brightness = False,
                         random_contrast = False,
                         random_saturation = False,
                         random_hue = False,
                         rotation_range = 0)

train_gen = flow_from_dataframe(core_idg, train_df,
                                path_col = 'path',
                                y_col = 'level_cat',
                                batch_size = batch_size)

valid_gen = flow_from_dataframe(valid_idg, valid_df,
                                path_col = 'path',
                                y_col = 'level_cat',
                                batch_size = batch_size)
```

Figure 6: Preprocessing - 2

In this phase of preprocessing multiple different techniques were implemented so that all images are consistent throughout.

6.3 Modeling

```python
from keras.applications.vgg16 import VGG16 as PTModel
from keras.applications.inception_resnet_v2 import InceptionResNetV2 as PTModel
from keras.applications.inception_v3 import InceptionV3 as PTModel
from keras.layers import (GlobalAveragePooling2D, Dense, Dropout, Input, Conv2D,
                          multiply, Lambda, BatchNormalization)
from keras.models import Model
import numpy as np

# Input layer
in_lay = Input(shape=t_x.shape[1:])

# Pre-trained model as the base
base_pretrained_model = PTModel(input_shape=t_x.shape[1:], include_top=False, weights='imagenet')
base_pretrained_model.trainable = False
pt_depth = base_pretrained_model.output_shape[-1]
pt_features = base_pretrained_model(in_lay)

# Batch normalization layer
bn_features = BatchNormalization()(pt_features)

# Attention mechanism to turn pixels in the GAP on and off
attn_layer = Conv2D(64, kernel_size=(1, 1), padding='same', activation='relu')(Dropout(0.5)(bn_features))
attn_layer = Conv2D(16, kernel_size=(1, 1), padding='same', activation='relu')(attn_layer)
attn_layer = Conv2D(8, kernel_size=(1, 1), padding='same', activation='relu')(attn_layer)
attn_layer = Conv2D(1, kernel_size=(1, 1), padding='valid', activation='sigmoid')(attn_layer)

# Fan it out to all of the channels
up_c2_w = np.ones((1, 1, 1, pt_depth))
up_c2 = Conv2D(pt_depth, kernel_size=(1, 1), padding='same', activation='linear', use_bias=False, weights=[up_c2_w])
up_c2.trainable = False
attn_layer = up_c2(attn_layer)

# Masking and pooling features
mask_features = multiply([attn_layer, bn_features])
gap_features = GlobalAveragePooling2D()(mask_features)
gap_mask = GlobalAveragePooling2D()(attn_layer)

# Rescaling
gap = Lambda(lambda x: x[0] / x[1], name='RescaleGAP')([gap_features, gap_mask])
gap_dr = Dropout(0.25)(gap)
dr_steps = Dropout(0.25)(Dense(128, activation='relu')(gap_dr))
out_layer = Dense(t_y.shape[-1], activation='softmax')(dr_steps)

# Creating the model
retina_model = Model(inputs=[in_lay], outputs=[out_layer])

# Top-2 accuracy metric
def top_2_accuracy(in_gt, in_pred):
    return top_k_categorical_accuracy(in_gt, in_pred, k=2)

# Compiling the model
retina_model.compile(optimizer='adam',
                     loss='categorical_crossentropy',
                     metrics=['categorical_accuracy', top_2_accuracy])

# Display model summary
retina_model.summary()
```

Figure 7: Model Implementation

The above code snippet shows the model-building process of the Inception ResNet V2 Szegedy et al. (2017), this was the model that was implemented for the classification purpose.

6.4 Evaluation

```python
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Assuming you have ground truth labels for your test images
true_labels = valid_df['grade']  # List of true labels corresponding to test images

# Convert true_labels to a numpy array
true_labels = np.array(true_labels)

# Get the predicted class indices from predictions
predicted_class_indices = np.argmax(predictions, axis=1)

print("Shape of true_labels:", true_labels.shape)
print("Shape of predicted_class_indices:", predicted_class_indices.shape)

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
label_encoder.fit(valid_df['grade'])
true_labels_encoded = label_encoder.transform(valid_df['grade'])

print("Encoded True Labels:", true_labels_encoded)


# Print classification report
print("Classification Report:")
print(classification_report(true_labels, predicted_class_indices))

# Print confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_class_indices)
print("Confusion Matrix:")
print(conf_matrix)

import seaborn as sns
from sklearn.metrics import confusion_matrix
sns.heatmap(conf_matrix,
            annot=True, fmt="d", cbar=False, cmap=plt.cm.Blues)
```

Figure 8: Evaluation Matrix Generation

The above code snippet generates an evaluation rubric that helps in evaluating the implemented model so that we can have a basis for the comparison.

7 Edge Methodology

Below are a few code snippets of the implementation of edge methodology.

```python
import cv2
import pandas as pd
from pathlib import Path

def calculate_edges(image_path):
    # Read the image
    image = cv2.imread(image_path)
    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Apply Canny edge detection
    edges = cv2.Canny(gray, 100, 200)
    return edges

# Apply the function to each row and store the result in a new column
retina_train_df['edges'] = retina_train_df['path'].apply(lambda x: calculate_edges(x))

# If you need to save the edge images as files and store the file paths
def save_edge_image(image_path):
    edges = calculate_edges(image_path)
    # Create a Path object from the image path
    path_obj = Path(image_path)
    # Construct the new path with '_edges' suffix before the file extension
    new_path = path_obj.with_name(path_obj.stem + '_edges' + path_obj.suffix)
    # Save the image
    cv2.imwrite(str(new_path), edges)
    return str(new_path)

# Then apply this function to your DataFrame
retina_train_df['edge_image_path'] = retina_train_df['path'].apply(lambda x: save_edge_image(x))
```

Figure 9: Getting Patterns of Edges

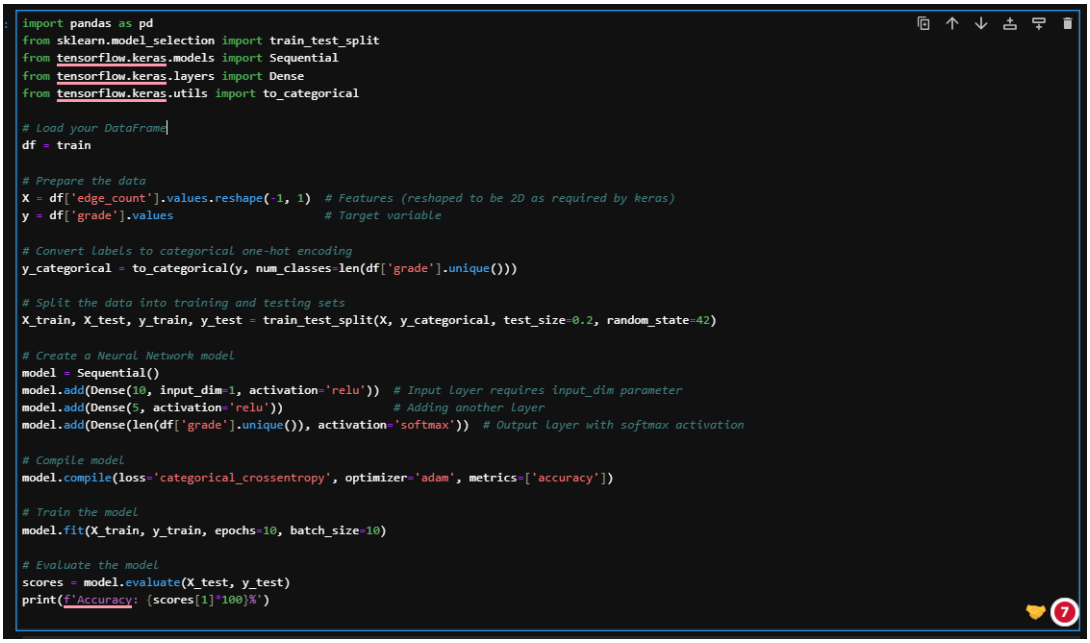
```python
import cv2
import numpy as np
from pathlib import Path

def count_edges(image_path):
    # Read the image
    image = cv2.imread(image_path)
    # Convert to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Apply Canny edge detection
    edges = cv2.Canny(gray, 100, 200)
    # Count the number of non-zero pixels (i.e., edge pixels)
    edge_count = np.count_nonzero(edges)
    return edge_count

# Apply the function to each row and store the result in a new column
retina_train_df['edge_count'] = retina_train_df['path'].apply(lambda x: count_edges(x))
```

```python
retina_train_df.head()
```

Figure 10: Getting Edge Count and Storing it



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

# Load your DataFrame
df = train

# Prepare the data
X = df['edge_count'].values.reshape(-1, 1)  # Features (reshaped to be 2D as required by keras)
y = df['grade'].values                      # Target variable

# Convert labels to categorical one-hot encoding
y_categorical = to_categorical(y, num_classes=len(df['grade'].unique()))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_categorical, test_size=0.2, random_state=42)

# Create a Neural Network model
model = Sequential()
model.add(Dense(10, input_dim=1, activation='relu'))  # Input layer requires input_dim parameter
model.add(Dense(5, activation='relu'))                # Adding another layer
model.add(Dense(len(df['grade'].unique()), activation='softmax'))  # Output layer with softmax activation

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

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=10)

# Evaluate the model
scores = model.evaluate(X_test, y_test)
print(f'Accuracy: {scores[1]*100}%')
```

Figure 11: Edge Methodology Modle

References

Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning, *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.