Configuration Manual

MSc Research Project
Data Analytics

Namrata Shrishail Tarade
Student ID: X22238867

School of Computing
National College of Ireland

Supervisor: Vikas Tomer

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Namrata Shrishail Tarade…………………………………………………………………………
Student ID:	X22238867……………………………………………………………………………………….……
Programme:	… MSc Data Analytics ……………………………… **Year:** ………2024……….
Module:	…MSc Research Project………………………………………………………………….………
Lecturer:	……Vikas Tomer…………………………………………………………………………….………
Submission Due Date:	…16th September 2024……………………………………………………………….………
Project Title:	A Novel Deep Learning Framework for Diabetic Retinopathy Detection IntegratingBen Graham and CLAHE Preprocessing.
Word Count:	…1436…………………………… **Page Count:** ………9……………………….…….………

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: ……Namrata Shrishail Tarade……………………………………………………

Date: ……16th September 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

Namrata Shrishail Tarade
Student ID: X22238867

1 Introduction

The project aims on detecting diabetic retinopathy from retinal images using Preprocessing techniques Ben Graham and CLAHE with different deep learning models that include customized CNN, EfficientNetB7, and NASNetMobile. The main target is to assess images of retinas into five categories of the severity of the disease diabetic retinopathy. It also describes the environment setup, data processing, model training, and evaluation processes required to achieve the results.

2 System Configuration

This project was done using Google Collaboratory. As for Python scripts, Google offers free cloud servers; yet there are certain conditions. Though Google Collab Pro, this feature can be used with more RAM and GPU usage. It incorporates an Intel(R) Core (TM) i5-1035G1 processor with CPU from 1. 00GHz. At the Collaboratory, the system specification that was used in this project entails an Intel (R) Intel(R) CPU, has the frequency of 19GHz, a Tesla T4 GPU with 2496 cores, 12GB of DDR5 VRAM, 30GB disk, RAM of 13GB. Other Libraries: Some of them are OpenCV, Numeric Python NumericPy, Data manipulation- Pandas, Data visualization- Plotly, Seaborn, Matplotlib, Image augmentation – ImgAug.

3 Data Collection

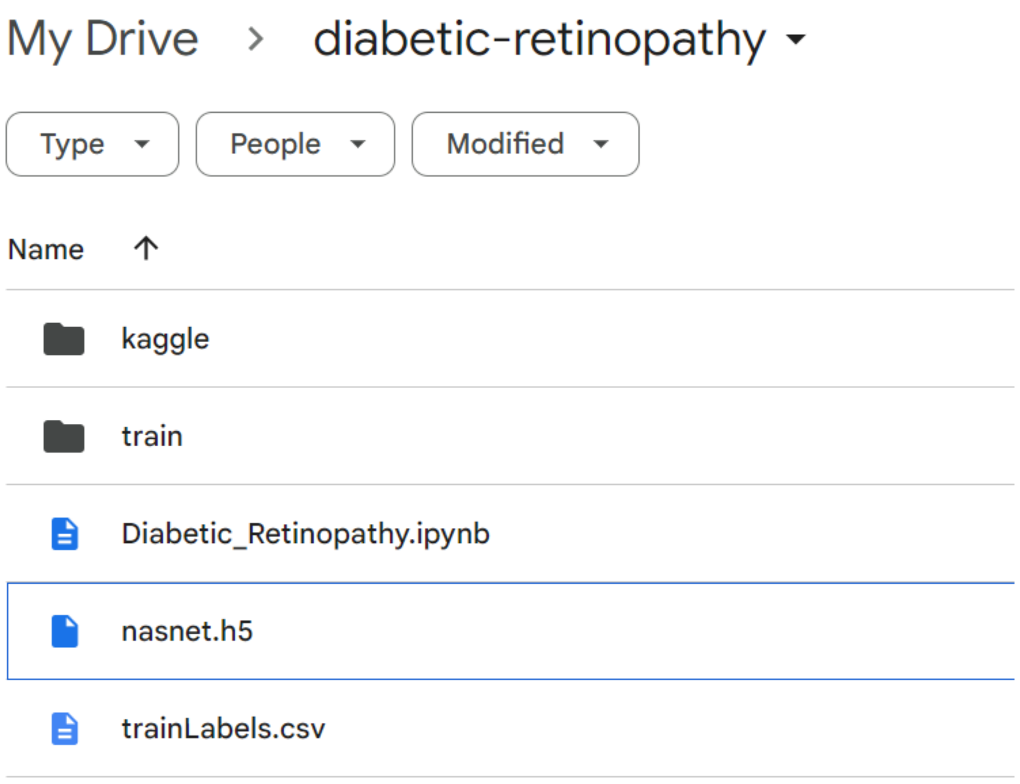
For this project, the dataset adopted is of retinal images which are categorized depending on the level of DR. The data used in this study was collected from the Kaggle competition called Diabetic Retinopathy Detection. The dataset can be accessed on this link[1].The dataset contains images, each associated with five classes:

• No_DR
• Mild
• Moderate
• Severe
• Proliferate_DR

4 Setting up Environment

[1] https://www.kaggle.com/c/diabetic-retinopathy-detection/data.

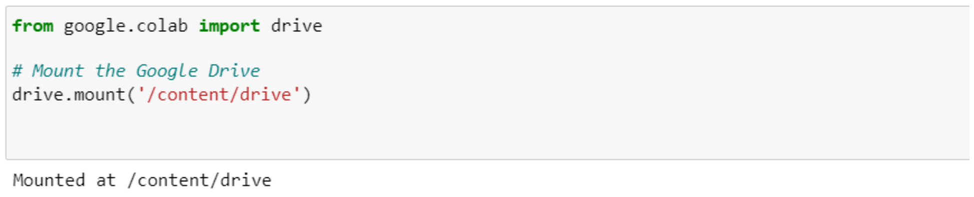
After downloading and unzipping the data from Kaggle, it was uploaded to Google Drive so that it will run on any machine. It eliminates the condition that limits the actual machine required. As shown in the below Figure 1, the data was uploaded.



```
kaggle
train
Diabetic_Retinopathy.ipynb
nasnet.h5
trainLabels.csv
```

Fig.1 Saved in the Google Drive

The dataset was additionally loaded to the Google Colab in order to utilize it for the project. Whenever you try to start the google disk you will get a message to verify your account to access the google disk by utilizing google collab as a security check. It is also worth to try to change the runtime environment to GPU since our data is in images which could help in increasing the performance. Python is used, and Google Collab comes with a Jupyter notebook setting already set up.



```python
from google.colab import drive

# Mount the Google Drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

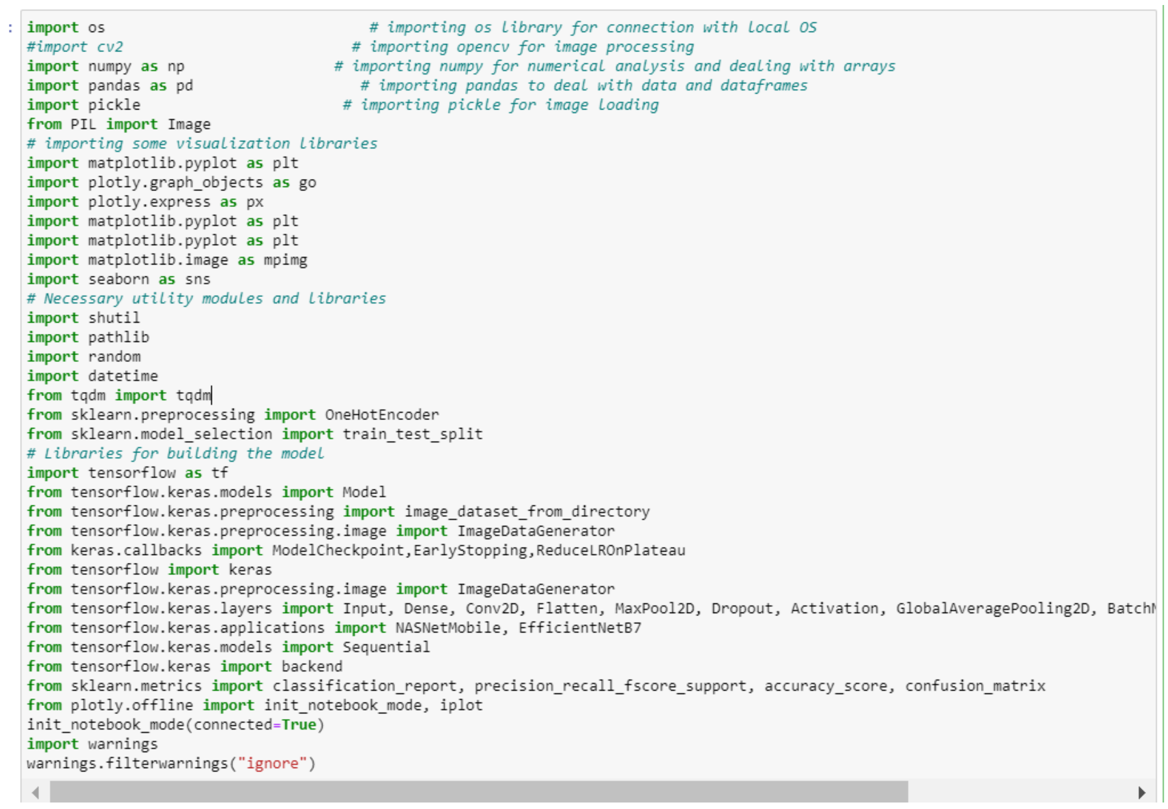
Fig.2 Mounting Drive

5 Data preprocessing

5.1 Python Libraries

At the start of the notebook, all necessary libraries for the data processing and modeling were imported. The critical libraries used include:

- Opencv for image processing
- The package used for the handling of dataframes are the pandas.
- Tensorflow for creating and training of models

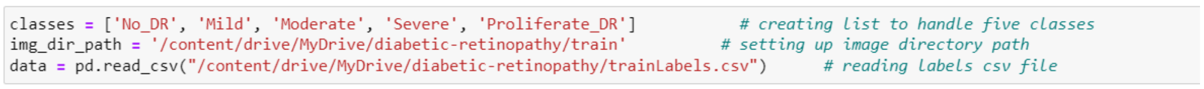


```
: import os                          # importing os library for connection with local OS
  #import cv2                       # importing opencv for image processing
  import numpy as np               # importing numpy for numerical analysis and dealing with arrays
  import pandas as pd                 # importing pandas to deal with data and dataframes
  import pickle                     # importing pickle for image loading
  from PIL import Image
  # importing some visualization libraries
  import matplotlib.pyplot as plt
  import plotly.graph_objects as go
  import plotly.express as px
  import matplotlib.pyplot as plt
  import matplotlib.pyplot as plt
  import matplotlib.image as mpimg
  import seaborn as sns
  # Necessary utility modules and libraries
  import shutil
  import pathlib
  import random
  import datetime
  from tqdm import tqdm
  from sklearn.preprocessing import OneHotEncoder
  from sklearn.model_selection import train_test_split
  # Libraries for building the model
  import tensorflow as tf
  from tensorflow.keras.models import Model
  from tensorflow.keras.preprocessing import image_dataset_from_directory
  from tensorflow.keras.preprocessing.image import ImageDataGenerator
  from keras.callbacks import ModelCheckpoint,EarlyStopping,ReduceLROnPlateau
  from tensorflow import keras
  from tensorflow.keras.preprocessing.image import ImageDataGenerator
  from tensorflow.keras.layers import Input, Dense, Conv2D, Flatten, MaxPool2D, Dropout, Activation, GlobalAveragePooling2D, BatchN
  from tensorflow.keras.applications import NASNetMobile, EfficientNetB7
  from tensorflow.keras.models import Sequential
  from tensorflow.keras import backend
  from sklearn.metrics import classification_report, precision_recall_fscore_support, accuracy_score, confusion_matrix
  from plotly.offline import init_notebook_mode, iplot
  init_notebook_mode(connected=True)
  import warnings
  warnings.filterwarnings("ignore")
```

Fig.3 Importing Python Libraries

5.2 Data Loading

The images were placed in a directory train and the labels were extracted from a CSV file known as trainLabels. csv.

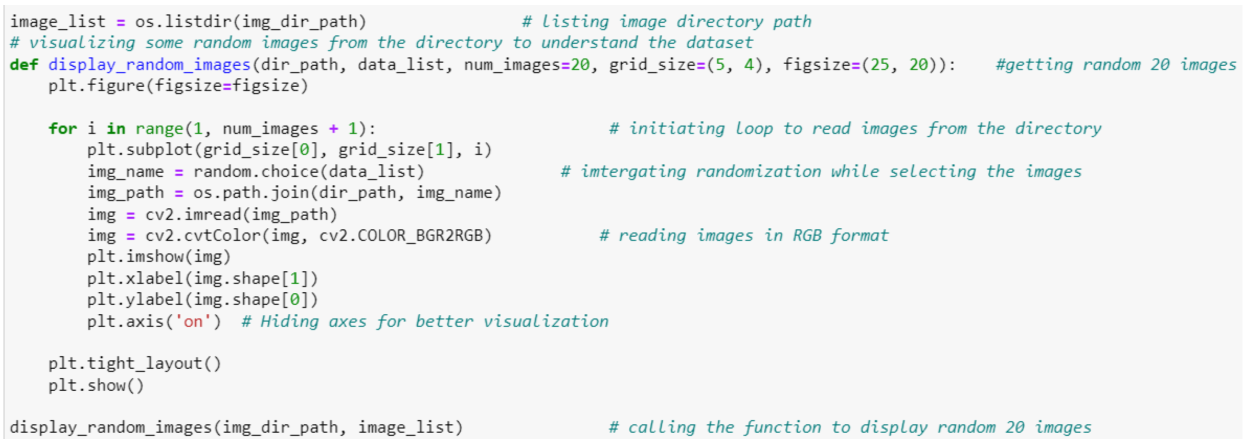


```
classes = ['No_DR', 'Mild', 'Moderate', 'Severe', 'Proliferate_DR']          # creating list to handle five classes
img_dir_path = '/content/drive/MyDrive/diabetic-retinopathy/train'          # setting up image directory path
data = pd.read_csv("/content/drive/MyDrive/diabetic-retinopathy/trainLabels.csv")       # reading labels csv file
```

Fig.4 Loading the data

5.3 Data Visualisation

In order to get a general idea about the type and quality of images used in the creation of the dataset a simple function is implemented which would select and display a random set of images along with their respective label.

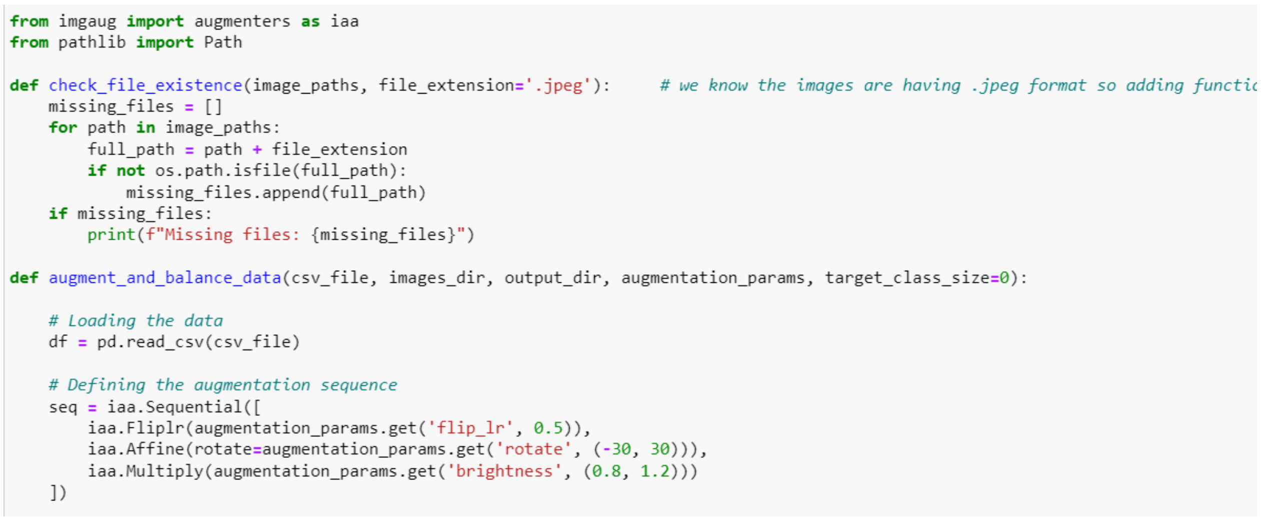


```
image_list = os.listdir(img_dir_path)                # listing image directory path
# visualizing some random images from the directory to understand the dataset
def display_random_images(dir_path, data_list, num_images=20, grid_size=(5, 4), figsize=(25, 20)):    #getting random 20 images
    plt.figure(figsize=figsize)

    for i in range(1, num_images + 1):                         # initiating loop to read images from the directory
        plt.subplot(grid_size[0], grid_size[1], i)
        img_name = random.choice(data_list)              # imtergating randomization while selecting the images
        img_path = os.path.join(dir_path, img_name)
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)           # reading images in RGB format
        plt.imshow(img)
        plt.xlabel(img.shape[1])
        plt.ylabel(img.shape[0])
        plt.axis('on')  # Hiding axes for better visualization

    plt.tight_layout()
    plt.show()

display_random_images(img_dir_path, image_list)              # calling the function to display random 20 images
```

Fig.5 Visualizing Random images

6 Data Augmentation and Transformation

To address the imbalance of the dataset, Data enhancement was performed this involves creating new data instances from existing instances but belonging to different classes. Borrowing the technology from the imgaug library, techniques such as horizontal flipping, rotation and brightness adjustment were done.



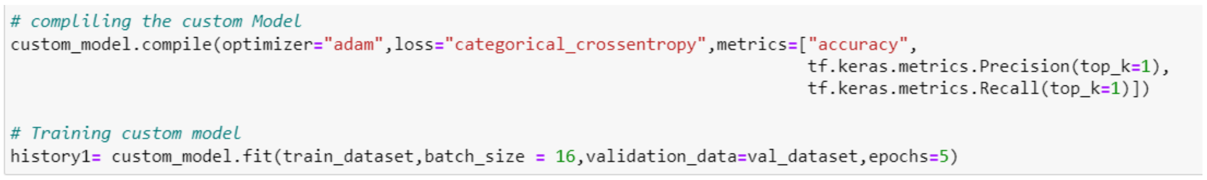
```
from imgaug import augmenters as iaa
from pathlib import Path

def check_file_existence(image_paths, file_extension='.jpeg'):     # we know the images are having .jpeg format so adding functi
    missing_files = []
    for path in image_paths:
        full_path = path + file_extension
        if not os.path.isfile(full_path):
            missing_files.append(full_path)
    if missing_files:
        print(f"Missing files: {missing_files}")

def augment_and_balance_data(csv_file, images_dir, output_dir, augmentation_params, target_class_size=0):

    # Loading the data
    df = pd.read_csv(csv_file)

    # Defining the augmentation sequence
    seq = iaa.Sequential([
        iaa.Fliplr(augmentation_params.get('flip_lr', 0.5)),
        iaa.Affine(rotate=augmentation_params.get('rotate', (-30, 30))),
        iaa.Multiply(augmentation_params.get('brightness', (0.8, 1.2)))
    ])
```

Fig.6 Data Augmentation

7 Model Design and Training

7.1 Custom CNN Model

Based on the architecture of deep convolutional neural networks, a custom CNN was created with several convolutional layers after which max-pooling and batch normalization layers were applied. This model was trained with categorical cross-entropy as the loss function and the Adam optimizer.

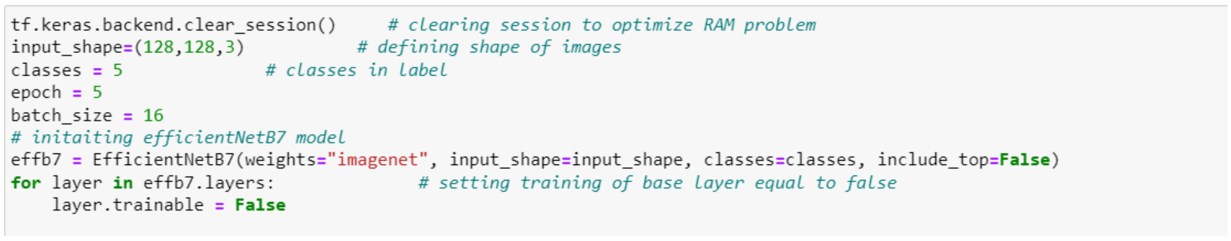


```
# compliling the custom Model
custom_model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy",
                                                                               tf.keras.metrics.Precision(top_k=1),
                                                                               tf.keras.metrics.Recall(top_k=1)])

# Training custom model
history1= custom_model.fit(train_dataset,batch_size = 16,validation_data=val_dataset,epochs=5)
```

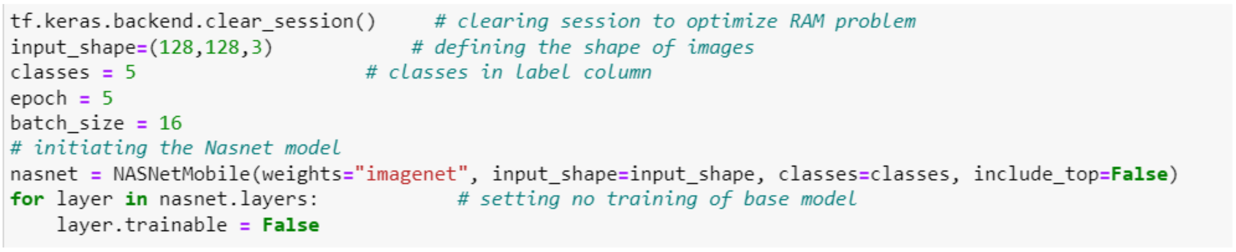
Fig.7 CNN Model

7.2 Transfer Learning Models

Two transfer learning models, EfficientNetB7 and NASNetMobile, were implemented:
• EfficientNetB7: Extra Dense layer + dropout and fine-tuned.
• NASNetMobile: As EfficientNetB7 but fine-tuned with additional dense and dropout layer.
Both models were trained up to 5 epochs and the learning rate was set to be reduced according to the plateaus on the validation accuracy using ReduceLROnPlateau.



```
tf.keras.backend.clear_session()     # clearing session to optimize RAM problem
input_shape=(128,128,3)          # defining shape of images
classes = 5             # classes in label
epoch = 5
batch_size = 16
# initaiting efficientNetB7 model
effb7 = EfficientNetB7(weights="imagenet", input_shape=input_shape, classes=classes, include_top=False)
for layer in effb7.layers:             # setting training of base layer equal to false
    layer.trainable = False
```

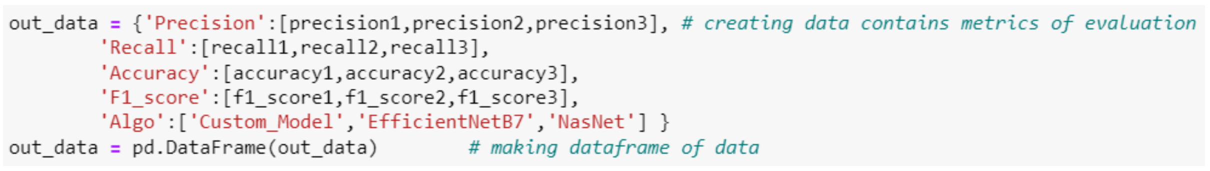


```
tf.keras.backend.clear_session()     # clearing session to optimize RAM problem
input_shape=(128,128,3)            # defining the shape of images
classes = 5                  # classes in label column
epoch = 5
batch_size = 16
# initiating the Nasnet model
nasnet = NASNetMobile(weights="imagenet", input_shape=input_shape, classes=classes, include_top=False)
for layer in nasnet.layers:            # setting no training of base model
    layer.trainable = False
```

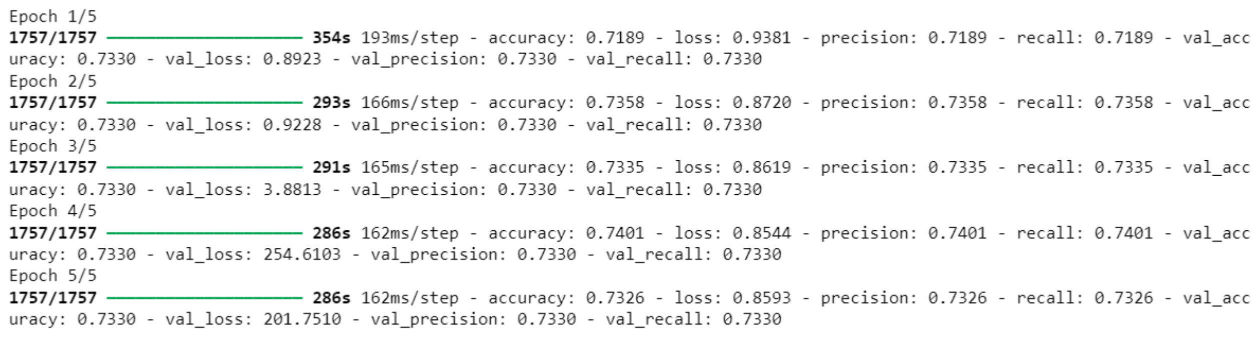
Fig.7.1 Transfer Learning Models

7.3 Model Training

The training stage is one of the main levels of the machine learning pipeline, at which the chosen models detect patterns in the data associated with various classes of diabetic retinopathy. Training consists of providing the models with data that are supposed to be labeled in such a way that the internal parameters associated with the model or known as weights need to adapt to the labeled data so as to reduce the gap between the labeled results and the results generated by the model. This section gives details on the training procedures for three models: first, a Custom CNN model, second, an EfficientNetB7 and, third, NASNetMobile. Every model was trained on the preprocessed dataset, and there are particular procedures oriented toward enhancing their effectiveness.



```
out_data = {'Precision':[precision1,precision2,precision3], # creating data contains metrics of evaluation
        'Recall':[recall1,recall2,recall3],
        'Accuracy':[accuracy1,accuracy2,accuracy3],
        'F1_score':[f1_score1,f1_score2,f1_score3],
        'Algo':['Custom_Model','EfficientNetB7','NasNet'] }
out_data = pd.DataFrame(out_data)        # making dataframe of data
```

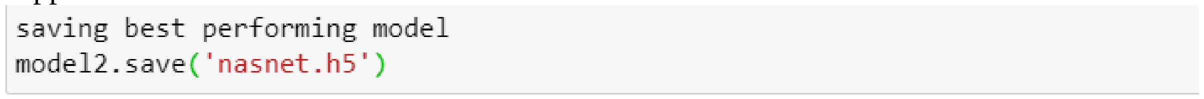


```
Epoch 1/5
1757/1757 ━━━━━━━━━━━━━━━━━━━━ 354s 193ms/step - accuracy: 0.7189 - loss: 0.9381 - precision: 0.7189 - recall: 0.7189 - val_acc
uracy: 0.7330 - val_loss: 0.8923 - val_precision: 0.7330 - val_recall: 0.7330
Epoch 2/5
1757/1757 ━━━━━━━━━━━━━━━━━━━━ 293s 166ms/step - accuracy: 0.7358 - loss: 0.8720 - precision: 0.7358 - recall: 0.7358 - val_acc
uracy: 0.7330 - val_loss: 0.9228 - val_precision: 0.7330 - val_recall: 0.7330
Epoch 3/5
1757/1757 ━━━━━━━━━━━━━━━━━━━━ 291s 165ms/step - accuracy: 0.7335 - loss: 0.8619 - precision: 0.7335 - recall: 0.7335 - val_acc
uracy: 0.7330 - val_loss: 3.8813 - val_precision: 0.7330 - val_recall: 0.7330
Epoch 4/5
1757/1757 ━━━━━━━━━━━━━━━━━━━━ 286s 162ms/step - accuracy: 0.7401 - loss: 0.8544 - precision: 0.7401 - recall: 0.7401 - val_acc
uracy: 0.7330 - val_loss: 254.6103 - val_precision: 0.7330 - val_recall: 0.7330
Epoch 5/5
1757/1757 ━━━━━━━━━━━━━━━━━━━━ 286s 162ms/step - accuracy: 0.7326 - loss: 0.8593 - precision: 0.7326 - recall: 0.7326 - val_acc
uracy: 0.7330 - val_loss: 201.7510 - val_precision: 0.7330 - val_recall: 0.7330
```

Fig.7.2 Model Training

7.4 Saving and Loading Model

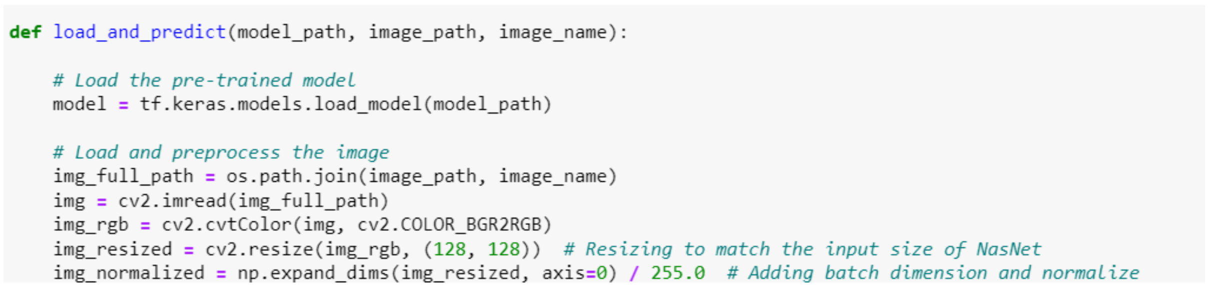
Model saving is an essential process in machine learning since the final model should be optimized for the specific task with all the hyperparameters fine-tuned. It will ultimately save the model in the. h5 format, which is a format familiar for saving Keras' models. This format ensures not only the existing architecture of the model but also the estimates that were achieved during training the model and that allows loading this model and further application it to predict. The save () Keras method is utilized in order to save the entire model structure, the optimizer condition, as well as the weights acquired during training. The following code snippet shows how the NASNetMobile model was saved:



```
saving best performing model
model2.save('nasnet.h5')
```

Fig 7.3 Model Saving

After saving, the model it is reinitiated to make more predictions from the testing data. The load_model () function from a Keras library is used to load the model from the saved. h5 file. This function is used to bring the model back to whichever state was present at end of training process, inclusive of weights and layers.

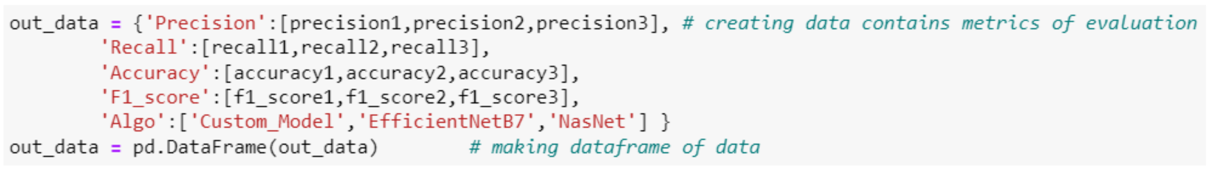


```
def load_and_predict(model_path, image_path, image_name):

    # Load the pre-trained model
    model = tf.keras.models.load_model(model_path)

    # Load and preprocess the image
    img_full_path = os.path.join(image_path, image_name)
    img = cv2.imread(img_full_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img_rgb, (128, 128))  # Resizing to match the input size of NasNet
    img_normalized = np.expand_dims(img_resized, axis=0) / 255.0  # Adding batch dimension and normalize
```

Fig 7.4 Loading Model

8 Model Evaluation

After training was done, the next step was to assess the performance of the developed trained models on test data set. Thus, out of all the models, the NASNetMobile was found to be the most efficient model, it offered a high balance between accuracy, precision, recall, and the F1 score. Since the goal of this model was to make predictions in subsequent applications. This process enables the identification of the models' accuracy in unseen data sets and the ease of understanding their performance in real life. The traditional metrics evaluation was made according to the performance indicators consisting of F1 score, precision, recall, and accuracy. These metrics offer a single view of every model and as to how good or bad it is especially in a multi class classification problem such as the detection of diabetic retinopathy.



```
out_data = {'Precision':[precision1,precision2,precision3], # creating data contains metrics of evaluation
        'Recall':[recall1,recall2,recall3],
        'Accuracy':[accuracy1,accuracy2,accuracy3],
        'F1_score':[f1_score1,f1_score2,f1_score3],
        'Algo':['Custom_Model','EfficientNetB7','NasNet'] }
out_data = pd.DataFrame(out_data)        # making dataframe of data
```

Fig.8 Comparative Analysis of Model