Configuration Manual

MSc Research Project
Data Analytics

Rahul Suryakant Vaydande
Student ID: x20181663

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rahul Suryakant Vaydande
Student ID:	x20181663
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Vladimir Milosavljevic
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	857
Page Count:	9

Signature:	Rahul Suryakant Vaydande
Date:	14th August 2022

Configuration Manual

Rahul Suryakant Vaydande
x20181663

1 Introduction

This study employs a hybrid convolutional neural network and LSTM to help detect diabetic retinopathy in retinal fundus scans. The research also implements two transfer learning model VGG19 and ResNet 50 which are trained on ImageNet dataset to evaluate the performance of my model. This configuration manual details all the processes that may be necessary for replication. The project design flow is described, from data gathering to model evaluation. As needed, further code snippets from other parts are also included.

2 System Configuration

Google Collaboratory was utilized to carry out the project's implementation. Google offers free cloud-based servers for running Python scripts, however there are certain restrictions. Google Collab Pro may be utilized with increased GPU and RAM use. The Google Collaboratory's system setup for this project includes an Intel(R) Xeon(R) CPU running at 2.00GHz, a Tesla T4 GPU with 2496 cores, 12Gigabytes of DDR5 VRAM, 30Gigabytes of accessible disk space, and 13GB of available RAM.

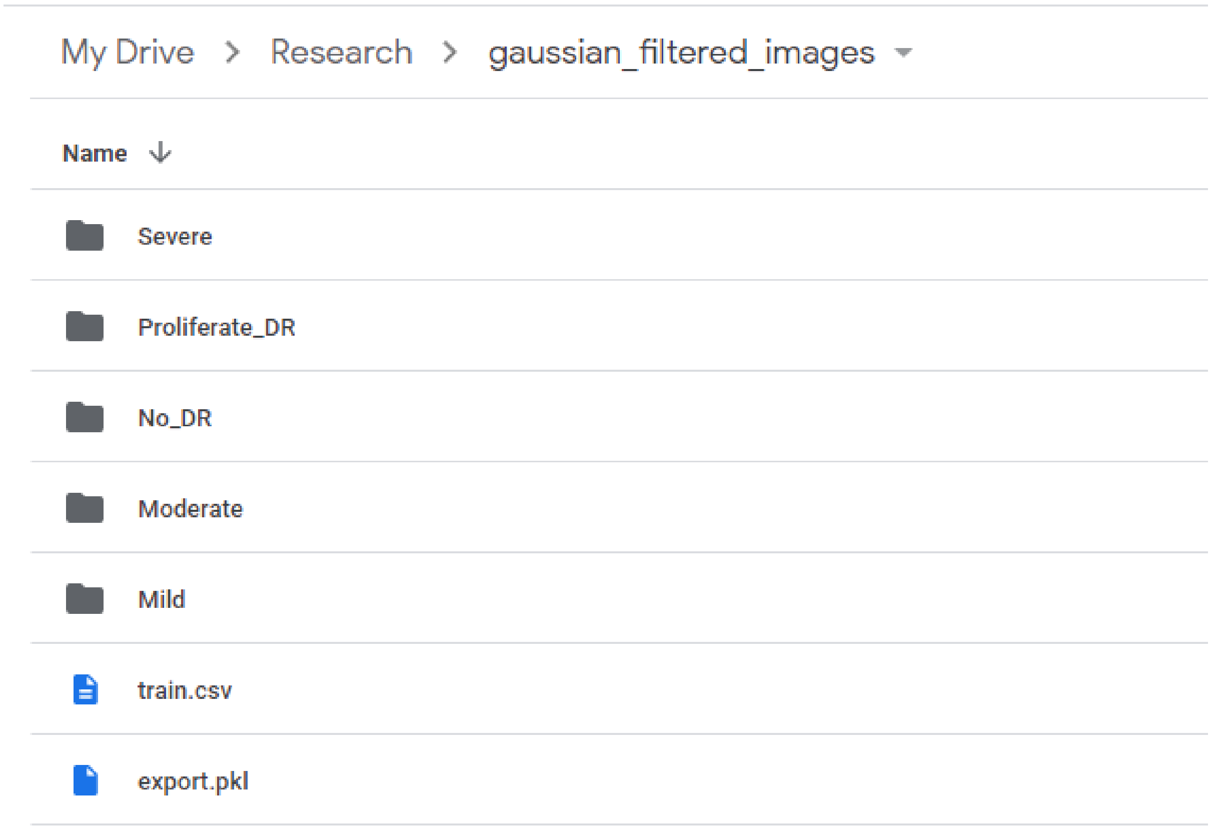
3 Data Collection

The dataset which is used for this research is sourced from an open-source website Kaggle. The dataset is basically a subset of the original dataset provided by APTOS 2019 Blindness Detection dataset. The dataset contains a total of 3662 files belonging to different classes whereas, the original dataset contains 88 GB of images. The dataset is available at link [1].

4 Environment Setup

The data was downloaded and unzipped from the kaggle and was uploaded on the google drive to make it executable on every machine. This removes the constraint on the machine requirement. The data was uploaded as shown in the below Figure 1.

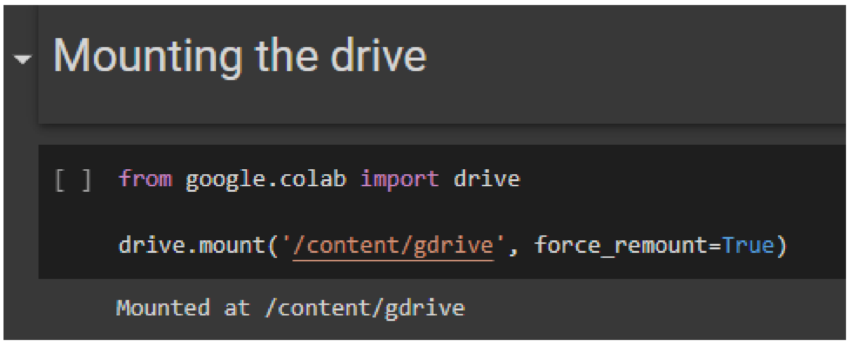
[1] `https://www.kaggle.com/datasets/sovitrath/diabetic-retinopathy-224x224-gaussian-filtered`


My Drive > Research > gaussian_filtered_images

Name ↓
Severe
Proliferate_DR
No_DR
Moderate
Mild
train.csv
export.pkl

Figure 1: Saved in Google Drive

The dataset was later connected to the google collab notebook to access it. For this we have used the predefined code for connecting the google drive provided by google as shown in the Figure 2 When mounting Google Disk, you will be prompted to verify your


Mounting the drive

```
[ ] from google.colab import drive

    drive.mount('/content/gdrive', force_remount=True)

    Mounted at /content/gdrive
```

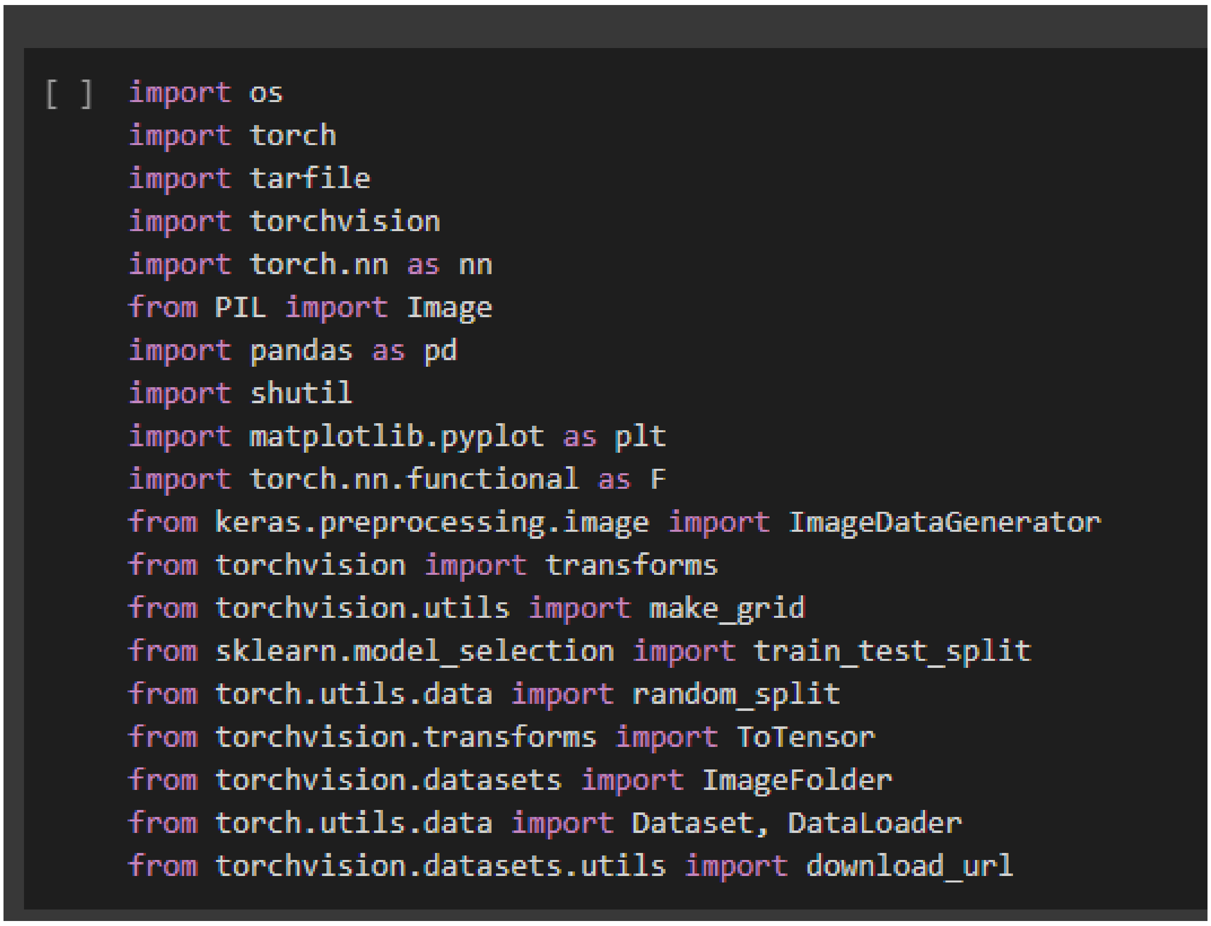
Figure 2: Mounting Drive

account in order to access the drive using Google Collab as part of their security check. Changing the runtime environment to GPU could help in improving performance as our data is in form of images. Python 3.9.3 is utilized for the coding, and Google Collab offers a pre-configured Jupyter notebook setup

5 Data Exportation

5.1 Importing Python Libraries

As shown in Figure 3, all of the required libraries were installed in Google Collab as shown below Figure 4



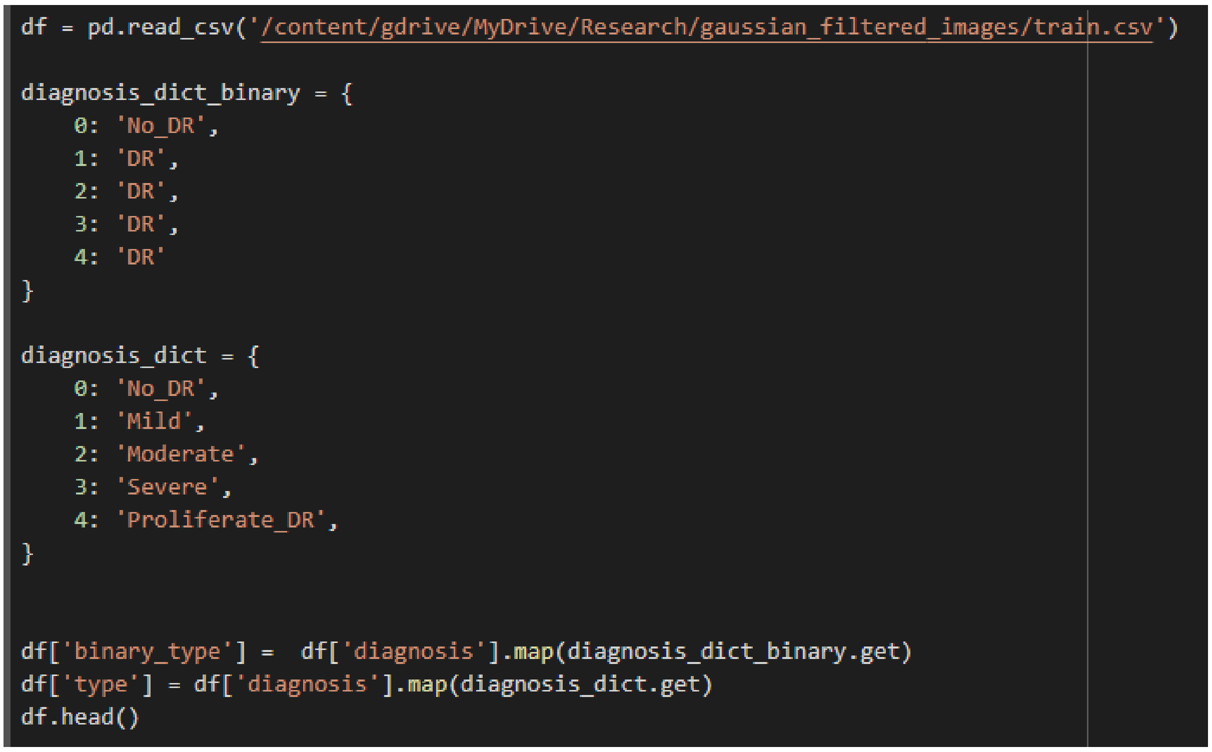
```
[ ]  import os
     import torch
     import tarfile
     import torchvision
     import torch.nn as nn
     from PIL import Image
     import pandas as pd
     import shutil
     import matplotlib.pyplot as plt
     import torch.nn.functional as F
     from keras.preprocessing.image import ImageDataGenerator
     from torchvision import transforms
     from torchvision.utils import make_grid
     from sklearn.model_selection import train_test_split
     from torch.utils.data import random_split
     from torchvision.transforms import ToTensor
     from torchvision.datasets import ImageFolder
     from torch.utils.data import Dataset, DataLoader
     from torchvision.datasets.utils import download_url
```

Figure 3: Importing Python Library

6 Exploratory Data Analysis

6.1 Verifying Class Distribution

When the dataset is in image form, the only EDA which we can perform is on the class distribution i.e., Binary Class and Multi Class distribution. The train.csv contains the information about the train dataset.

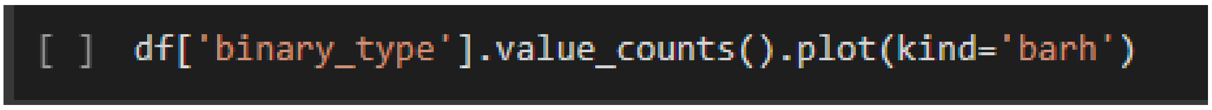


```
df = pd.read_csv('/content/gdrive/MyDrive/Research/gaussian_filtered_images/train.csv')

diagnosis_dict_binary = {
    0: 'No_DR',
    1: 'DR',
    2: 'DR',
    3: 'DR',
    4: 'DR'
}

diagnosis_dict = {
    0: 'No_DR',
    1: 'Mild',
    2: 'Moderate',
    3: 'Severe',
    4: 'Proliferate_DR',
}


df['binary_type'] =  df['diagnosis'].map(diagnosis_dict_binary.get)
df['type'] = df['diagnosis'].map(diagnosis_dict.get)
df.head()
```

Figure 4: Exploratory Data Analysis

6.1.1 DR/No-DR class distribution

This code was used for the getting insights about how many DR or NO DR images are present in the dataset.

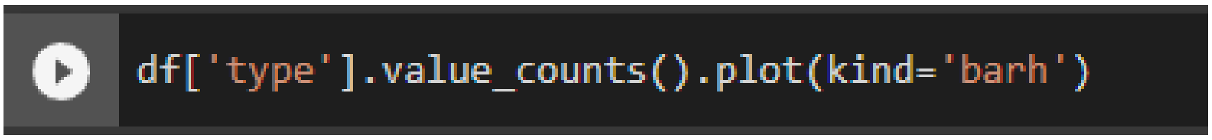


```
[ ]  df['binary_type'].value_counts().plot(kind='barh')
```

Figure 5: No-DR/DR Distribution

6.1.2 Multi-class distribution

This code was used for the getting insights about how the different classes of DR are present in the dataset.

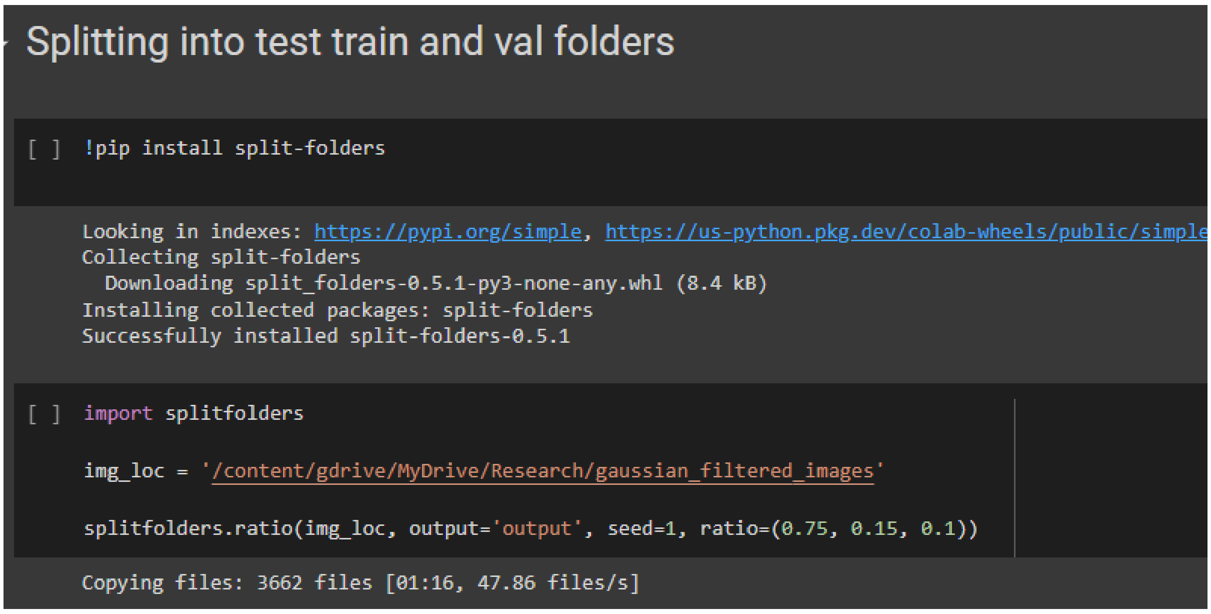


```
df['type'].value_counts().plot(kind='barh')
```

Figure 6: Multi-Class Distribution

7 Data Pre-processing

The dataset was split in train, test and validation using the split-folder python library which automatically splits the input datasets into different set into a output folder depending upon the ratio provided.



Splitting into test train and val folders

```
[ ] !pip install split-folders

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple
    Collecting split-folders
      Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
    Installing collected packages: split-folders
    Successfully installed split-folders-0.5.1

[ ] import splitfolders

    img_loc = '/content/gdrive/MyDrive/Research/gaussian_filtered_images'

    splitfolders.ratio(img_loc, output='output', seed=1, ratio=(0.75, 0.15, 0.1))

    Copying files: 3662 files [01:16, 47.86 files/s]
```

Figure 7: Data Splitting

8 Data Transformation

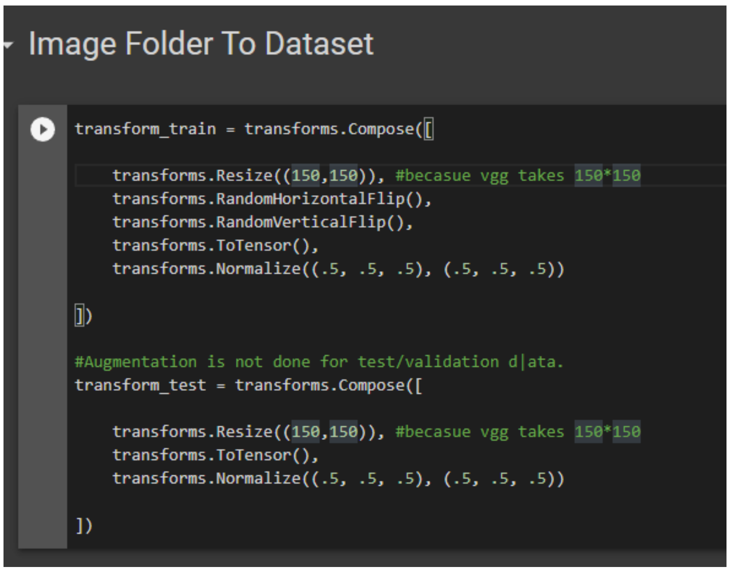


Image Folder To Dataset

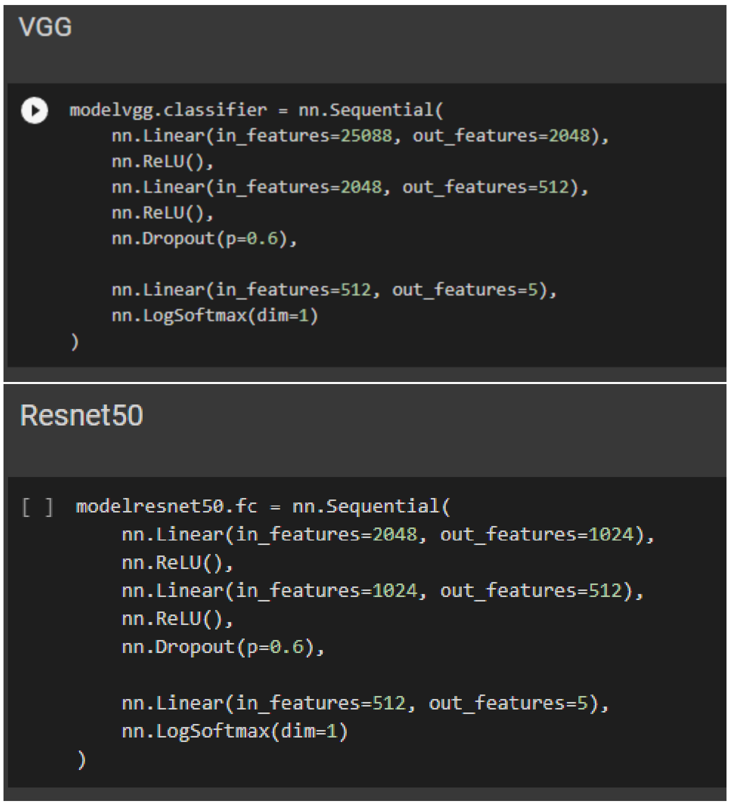
```
transform_train = transforms.Compose([

    transforms.Resize((150,150)), #becasue vgg takes 150*150
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((.5, .5, .5), (.5, .5, .5))

])

#Augmentation is not done for test/validation d|ata.
transform_test = transforms.Compose([

    transforms.Resize((150,150)), #becasue vgg takes 150*150
    transforms.ToTensor(),
    transforms.Normalize((.5, .5, .5), (.5, .5, .5))

])
```

Figure 8: Data Augmentation

After the data pre-processing, I performed the data transformation on train dataset which included steps such as random flips (Vertical and Horizontal), Normalization and Resizing because the VGG19 model works well if the image resolution is 150x150 and finally converted the images into tensors. The test and validation dataset were only resized and normalize.

9 Data Modelling

The transfer learning VGG19 and ResNet50 were downloaded, all the layers were frozen and the classifier layer was changed for classifying the DR images into 5 categories The


VGG

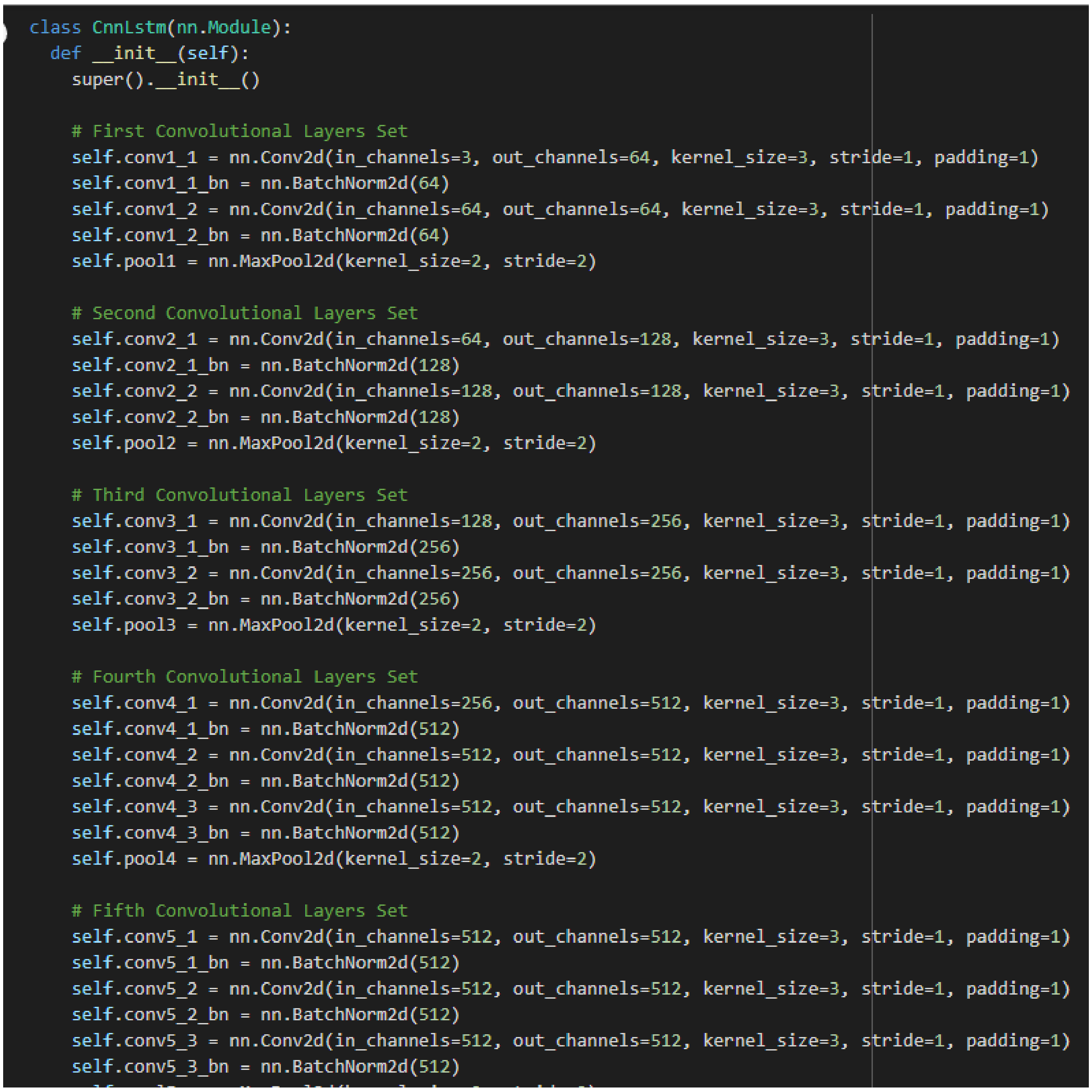
```
modelvgg.classifier = nn.Sequential(
    nn.Linear(in_features=25088, out_features=2048),
    nn.ReLU(),
    nn.Linear(in_features=2048, out_features=512),
    nn.ReLU(),
    nn.Dropout(p=0.6),

    nn.Linear(in_features=512, out_features=5),
    nn.LogSoftmax(dim=1)
)
```

Resnet50

```
[ ]  modelresnet50.fc = nn.Sequential(
         nn.Linear(in_features=2048, out_features=1024),
         nn.ReLU(),
         nn.Linear(in_features=1024, out_features=512),
         nn.ReLU(),
         nn.Dropout(p=0.6),

         nn.Linear(in_features=512, out_features=5),
         nn.LogSoftmax(dim=1)
     )
```

Figure 9: Data Modelling

CNN model with the LSTM as a classifier was designed in the research, the model consisted on 12 convolution layer, 5 max pooling layer, and 2 LSTM layer and 1 output layer. ReLU as an activation function for convolution layer was used.

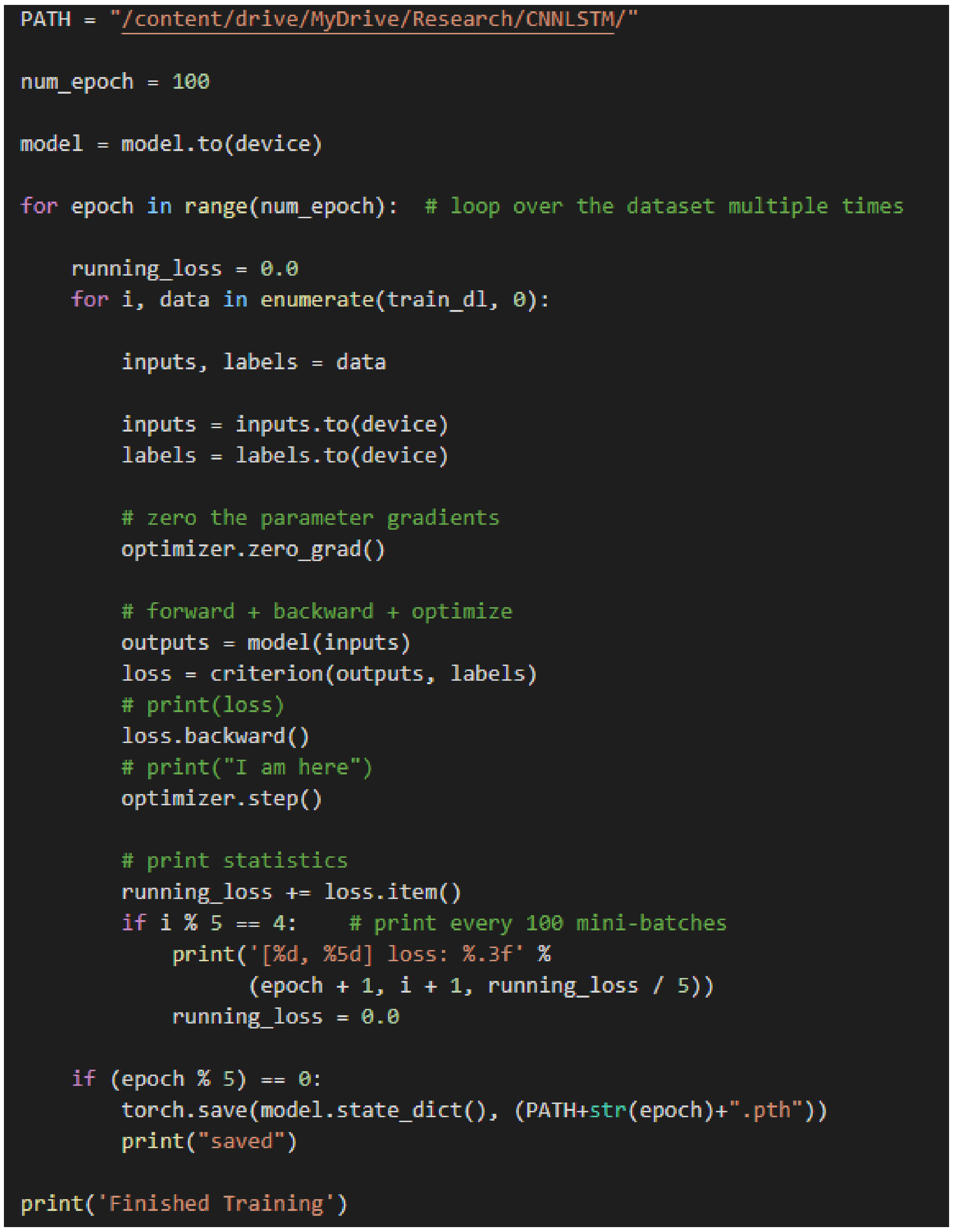


```
class CnnLstm(nn.Module):
  def __init__(self):
    super().__init__()

    # First Convolutional Layers Set
    self.conv1_1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=1, padding=1)
    self.conv1_1_bn = nn.BatchNorm2d(64)
    self.conv1_2 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1, padding=1)
    self.conv1_2_bn = nn.BatchNorm2d(64)
    self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

    # Second Convolutional Layers Set
    self.conv2_1 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1)
    self.conv2_1_bn = nn.BatchNorm2d(128)
    self.conv2_2 = nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, stride=1, padding=1)
    self.conv2_2_bn = nn.BatchNorm2d(128)
    self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

    # Third Convolutional Layers Set
    self.conv3_1 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, stride=1, padding=1)
    self.conv3_1_bn = nn.BatchNorm2d(256)
    self.conv3_2 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, stride=1, padding=1)
    self.conv3_2_bn = nn.BatchNorm2d(256)
    self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

    # Fourth Convolutional Layers Set
    self.conv4_1 = nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, stride=1, padding=1)
    self.conv4_1_bn = nn.BatchNorm2d(512)
    self.conv4_2 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, stride=1, padding=1)
    self.conv4_2_bn = nn.BatchNorm2d(512)
    self.conv4_3 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, stride=1, padding=1)
    self.conv4_3_bn = nn.BatchNorm2d(512)
    self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)

    # Fifth Convolutional Layers Set
    self.conv5_1 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, stride=1, padding=1)
    self.conv5_1_bn = nn.BatchNorm2d(512)
    self.conv5_2 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, stride=1, padding=1)
    self.conv5_2_bn = nn.BatchNorm2d(512)
    self.conv5_3 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, stride=1, padding=1)
    self.conv5_3_bn = nn.BatchNorm2d(512)
```

Figure 10: Data Modelling

10 Model Training

Each model was trained on the dataset the below function. Every 5th model was then saved into the drive and the model with the greatest accuracy and less loss was selected and loaded for the predicting on test and validation set.

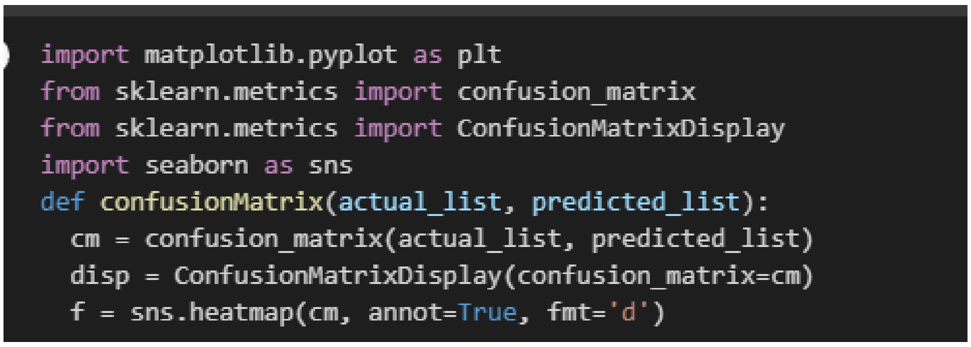


```
PATH = "/content/drive/MyDrive/Research/CNNLSTM/"

num_epoch = 100

model = model.to(device)

for epoch in range(num_epoch):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_dl, 0):

        inputs, labels = data

        inputs = inputs.to(device)
        labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        # print(loss)
        loss.backward()
        # print("I am here")
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 5 == 4:    # print every 100 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 5))
            running_loss = 0.0

    if (epoch % 5) == 0:
        torch.save(model.state_dict(), (PATH+str(epoch)+".pth"))
        print("saved")

print('Finished Training')
```

Figure 11: Model training

11 Model Evaluation

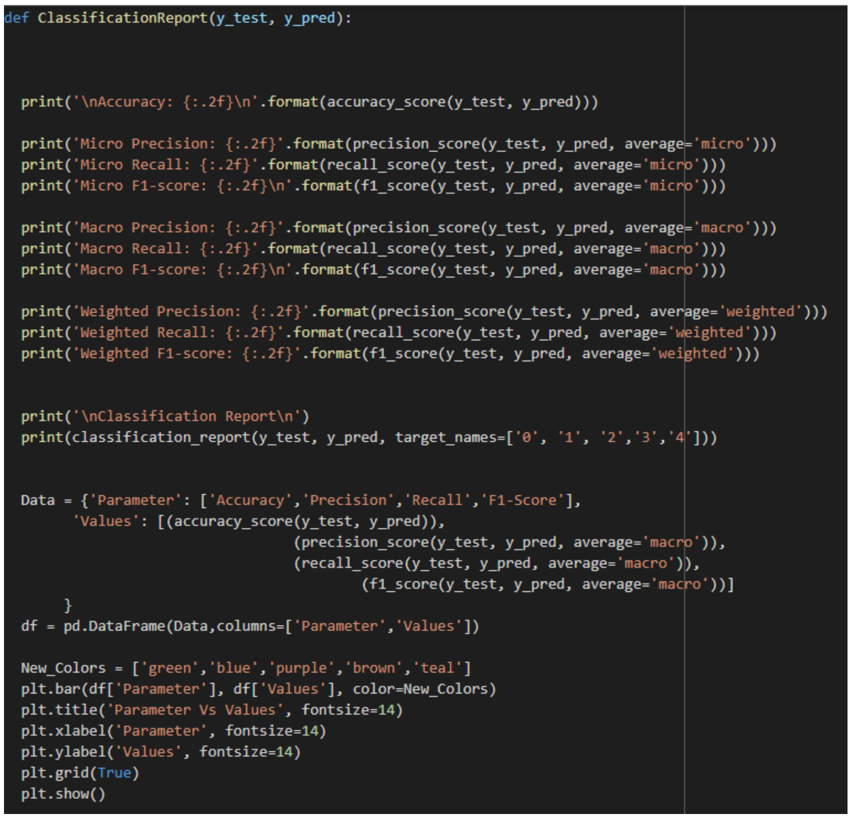
The models implemented in this research were evaluated based on the accuracy, precision, recall and F1 score as the traditional evaluation parameters doesn't satisfy. The confusion matrix was used to determine these values.



```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import seaborn as sns
def confusionMatrix(actual_list, predicted_list):
  cm = confusion_matrix(actual_list, predicted_list)
  disp = ConfusionMatrixDisplay(confusion_matrix=cm)
  f = sns.heatmap(cm, annot=True, fmt='d')
```

Figure 12: Confusion Matix Implementation

Other than the confusion matrix, I have used a classification report which provides the detailed report for the output of the model. The report consists of different performance parametes such as Micro, Macro and Weighted Precision, Recall and F1 score. The parametes can be also visually analyzed with the help of the bar graphs.



```python
def ClassificationReport(y_test, y_pred):



  print('\nAccuracy: {:.2f}\n'.format(accuracy_score(y_test, y_pred)))

  print('Micro Precision: {:.2f}'.format(precision_score(y_test, y_pred, average='micro')))
  print('Micro Recall: {:.2f}'.format(recall_score(y_test, y_pred, average='micro')))
  print('Micro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred, average='micro')))

  print('Macro Precision: {:.2f}'.format(precision_score(y_test, y_pred, average='macro')))
  print('Macro Recall: {:.2f}'.format(recall_score(y_test, y_pred, average='macro')))
  print('Macro F1-score: {:.2f}\n'.format(f1_score(y_test, y_pred, average='macro')))

  print('Weighted Precision: {:.2f}'.format(precision_score(y_test, y_pred, average='weighted')))
  print('Weighted Recall: {:.2f}'.format(recall_score(y_test, y_pred, average='weighted')))
  print('Weighted F1-score: {:.2f}'.format(f1_score(y_test, y_pred, average='weighted')))


  print('\nClassification Report\n')
  print(classification_report(y_test, y_pred, target_names=['0', '1', '2','3','4']))


  Data = {'Parameter': ['Accuracy','Precision','Recall','F1-Score'],
        'Values': [(accuracy_score(y_test, y_pred)),
                              (precision_score(y_test, y_pred, average='macro')),
                              (recall_score(y_test, y_pred, average='macro')),
                                    (f1_score(y_test, y_pred, average='macro'))]
       }
  df = pd.DataFrame(Data,columns=['Parameter','Values'])

  New_Colors = ['green','blue','purple','brown','teal']
  plt.bar(df['Parameter'], df['Values'], color=New_Colors)
  plt.title('Parameter Vs Values', fontsize=14)
  plt.xlabel('Parameter', fontsize=14)
  plt.ylabel('Values', fontsize=14)
  plt.grid(True)
  plt.show()
```

Figure 13: Classification Report Implementation

Multi-Classification: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826