Configuration Manual

MSc Research Project
Programme Name

Piyush Kishor Dhande
Student ID: x20115725

School of Computing
National College of Ireland

Supervisor: Prof. Noel Cosgrave

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Piyush Kishor Dhande
Student ID:	x20115725
Programme:	Data Analytics
Year:	2021-2022
Module:	MSc Research Project
Supervisor:	Prof. Noel Cosgrave
Submission Due Date:	31/01/2022
Project Title:	Image Classification of Melanoma SkinCancer using Deep Learning Approach
Word Count:	XXX
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:	31st January 2022

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

Piyush Kishor Dhande
x20115725

1 Introduction

This research project detects Skin cancer from dermatoscopic image using VGG16-CapsNet model and Transfer Learning methods. In this manual all the required steps starting from collection of data to implementation of model will be mentioned. Code snippets are used to explain step by step process.

2 System Configuration

The implementation part are divided into two parts.

- Processor : Intel Core i5-8200U
- Operating system : Windows 10
- Ram : 16GB DDR3
- Storage : 512 GB SSD

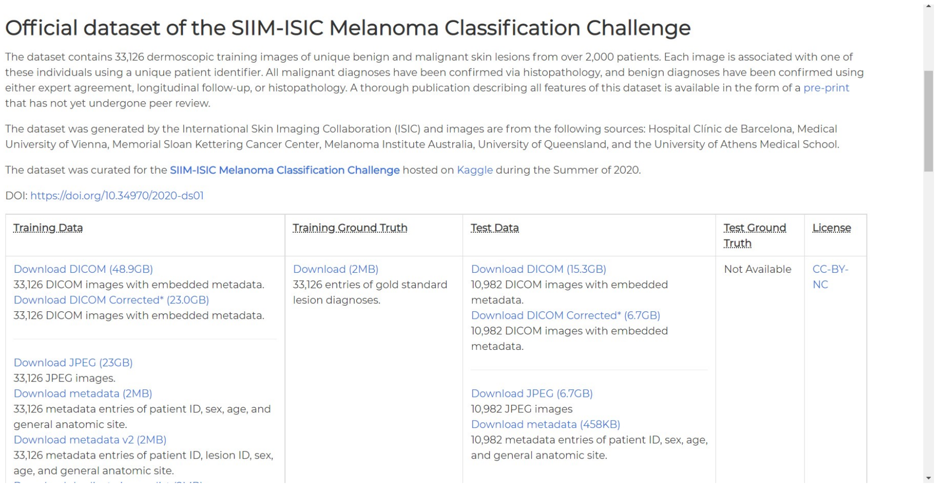
3 Software Requirement

- Anaconda Version: 4.8.3
- Kaggle Notebook
- Jupyter Notebook
- Spyder 3.7
- Google Chrome 96.0.4

4 Dataset Collection

The dataset used in this research was taken from ISIC archives[1].Dataset is available in 2 formats DICOM and JPEG, opt for JPEG training imageset along with its metadata. The dataset consists of 33,129 images of size 6000x4000 pixels.

[1] `https://challenge2020.isic-archive.com/`



Official dataset of the SIIM-ISIC Melanoma Classification Challenge

The dataset contains 33,126 dermoscopic training images of unique benign and malignant skin lesions from over 2,000 patients. Each image is associated with one of these individuals using a unique patient identifier. All malignant diagnoses have been confirmed via histopathology, and benign diagnoses have been confirmed using either expert agreement, longitudinal follow-up, or histopathology. A thorough publication describing all features of this dataset is available in the form of a pre-print that has not yet undergone peer review.

The dataset was generated by the International Skin Imaging Collaboration (ISIC) and images are from the following sources: Hospital Clínic de Barcelona, Medical University of Vienna, Memorial Sloan Kettering Cancer Center, Melanoma Institute Australia, University of Queensland, and the University of Athens Medical School.

The dataset was curated for the SIIM-ISIC Melanoma Classification Challenge hosted on Kaggle during the Summer of 2020.

DOI: https://doi.org/10.34970/2020-ds01

Training Data	Training Ground Truth	Test Data	Test Ground Truth	License
Download DICOM (48.9GB) 33,126 DICOM images with embedded metadata. Download DICOM Corrected* (23.0GB) 33,126 DICOM images with embedded metadata.	Download (2MB) 33,126 entries of gold standard lesion diagnoses.	Download DICOM (15.3GB) 10,982 DICOM images with embedded metadata. Download DICOM Corrected* (6.7GB) 10,982 DICOM images with embedded metadata.	Not Available	CC-BY-NC
Download JPEG (23GB) 33,126 JPEG images. Download metadata (2MB) 33,126 metadata entries of patient ID, sex, age, and general anatomic site. Download metadata v2 (2MB) 33,126 metadata entries of patient ID, lesion ID, sex, age, and general anatomic site.		Download JPEG (6.7GB) 10,982 JPEG images Download metadata (458KB) 10,982 metadata entries of patient ID, sex, age, and general anatomic site.		

Figure 1: Dataset source

5 Environmental Setup

The implementation part is divided into 2 parts

5.1 Part 1: Use of Local Environment

- In part 1 the Exploratory Data Analysis, Image Augmentation part was done on local system.
- Implementation of VGG-16 and ResNet-50 model

5.2 Part 2: Use of Kaggle Notebook

- In part 2 the training and testing of model is done on Kaggle Notebook.

5.3 Set up Kaggle Notebook

To set up a kaggle notebook it is mandatory to sign-up or log-in on Kaggle. From figure 6 create a new notebook on Kaggle.

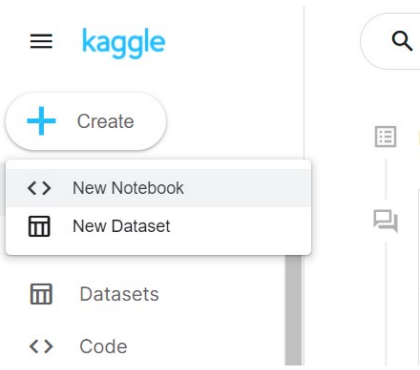


Figure 2: Kaggle notebook create

Dataset has to be uploaded from local system to Kaggle's cloud storage. To increase the performance the runtime environment can be changed to GPU. The python version used on Kaggle is 3.7. After uploading dataset the path can be used in code to import data.

6 Python Library

Numpy Version: 1.18.0 Pandas Version: 1.1.5 Tensorflow-gpu Version: 1.15.0 Matplotlib Version: 3.2.1 Sklearn Version: 0.22.2 Keras Version: 2.2.4 OpenCV Version: 4.1.2

7 Data load, split and pre-process

7.1 Pre-process

Importing dependencies and datasets and after that class labels are extracted along with image names https://github.com/bojone/Capsule/

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 18 14:43:17 2021

@author: dhand
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import os
import seaborn as sns
import tensorflow as tf
import random

from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_path = r"X:\\Research_Project\\ISIC_2020_Training_JPEG\\train\\"

test_path = r"X:\\Research_Project\\ISIC_2020_Test_JPEG\\ISIC_2020_Test_Input\\"

label_data = pd.read_csv("X:\\Research_Project\\ISIC_2020_Training_GroundTruth.csv")

save_dir_path = 'X:\\Research_Project\\ISIC_2020_Training_JPEG\\aug\\'

labels = ['benign', 'malignant']
filename_list = []
image_list = []
label_list = []


for filename in os.listdir(train_path):
    file_path = train_path+filename
    # print(file_path)
    # print(filename)
    imagename = filename[:-4]
    arr = label_data.query('image_name==@imagename')
    if len(arr) > 0:
        target = int(arr['target'])
        if target == 1:
            label_list.append(target)
            image_list.append(imagename)
            filename_list.append(file_path)
```

Figure 3: Image import

After importing images the data set is split into train(60%), test(20%) and validation set(20%) Due to class imbalance we only augmented malignant class and from benign class took 8000 images and segregated all images into train, test and validation folder

```
malignant_aug_images=[]

for file_name in os.listdir(save_dir_path):
    malignant_aug_images.append([save_dir_path+file_name,file_name[:-4]])

random.shuffle(malignant_aug_images)

train_malignant = malignant_aug_images[:int(len(malignant_aug_images)*0.6)]
val_malignant = malignant_aug_images[int(len(malignant_aug_images)*0.6):int(len(malignant_aug_images)*0.8)
test_malignant = malignant_aug_images[int(len(malignant_aug_images)*0.8):]

newImageSet_file_path='X:\\Research_Project\\new_imageset\\'

for item in train_malignant:
    os.rename(item[0],newImageSet_file_path+"train\\malignant\\"+item[1]+'.jpg')


for item in val_malignant:
    os.rename(item[0],newImageSet_file_path+"val\\malignant\\"+item[1]+'.jpg')


for item in test_malignant:
    os.rename(item[0],newImageSet_file_path+"test\\malignant\\"+item[1]+'.jpg')
```

Figure 4: Data augmentation for malignant class

```
benign_read_count = 8000
benign_images = []
i=0
for filename in os.listdir(train_path):
    if i>benign_read_count:
        break
    file_path = train_path+filename
    imagename = filename[:-4]
    arr = label_data.query('image_name==@imagename')
    if len(arr) > 0:
        target = int(arr['target'])
        if target == 0:
            i=i+1
            benign_images.append([file_path, imagename])

train_benign = benign_images[:int(len(benign_images)*0.6)]
val_benign = benign_images[int(len(benign_images)*0.6):int(len(benign_images)*0.8)]
test_benign = benign_images[int(len(benign_images)*0.8):]


for item in train_benign:
    os.rename(item[0],newImageSet_file_path+"train\\benign\\"+item[1]+'.jpg')


for item in val_benign:
    os.rename(item[0],newImageSet_file_path+"val\\benign\\"+item[1]+'.jpg')


for item in test_benign:
    os.rename(item[0],newImageSet_file_path+"test\\benign\\"+item[1]+'.jpg')
```

Figure 5: Benign class

7.2 Modelling of VGG16-CapsNet

The dependencies are imported for VGG16-CapsNet model[2] and library required for this file to run is

- !pip install tensorflow-gpu==1.15
- !pip install keras==2.2.4
- !pip install 'h5py==2.10.0' –force-reinstall

After importing data capsule function is defined Figure 8 After applying output of VGG16 as an input in capsule layer this summary is generated.

[2] `https://github.com/bojone/Capsule/`

```python
from __future__ import print_function
from keras import backend as K
from keras import activations
from keras import utils
from keras.models import Model
from keras.layers import *
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.optimizers import RMSprop, Adam, SGD, Nadam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
from keras import regularizers


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

IMG_SIZE = 299
```

Figure 6: Dependencies for VGG16-CapsNet model

7.3 Modelling of Transfer learning models

This modelling part is implemented on local machine. Here in Figure 9 dependencies are imported that are common for both models After this step summary of model ResNet is generated in Figure 10

```
class Capsule(Layer):

    def __init__(self,
                 num_capsule,
                 dim_capsule,
                 routings=3, # Test number of routing with (1, 2, 3, 4) - Default = 3
                 share_weights=True,
                 activation='squash',
                 **kwargs):
        super(Capsule, self).__init__(**kwargs)
        self.num_capsule = num_capsule
        self.dim_capsule = dim_capsule
        self.routings = routings
        self.share_weights = share_weights
        if activation == 'squash':
            self.activation = squash
        else:
            self.activation = activations.get(activation)

    def build(self, input_shape):
        input_dim_capsule = input_shape[-1]
        if self.share_weights:
            self.kernel = self.add_weight(
                name='capsule_kernel',
                shape=(1, input_dim_capsule,
                       self.num_capsule * self.dim_capsule),
                initializer='glorot_uniform',
                trainable=True)
        else:
            input_num_capsule = input_shape[-2]
            self.kernel = self.add_weight(
                name='capsule_kernel',
                shape=(input_num_capsule, input_dim_capsule,
                       self.num_capsule * self.dim_capsule),
                initializer='glorot_uniform',
                trainable=True)

    def call(self, inputs):

        if self.share_weights:
            hat_inputs = K.conv1d(inputs, self.kernel)
        else:
            hat_inputs = K.local_conv1d(inputs, self.kernel, [1], [1])

        batch_size = K.shape(inputs)[0]
        input_num_capsule = K.shape(inputs)[1]
        hat_inputs = K.reshape(hat_inputs,
                               (batch_size, input_num_capsule,
                                self.num_capsule, self.dim_capsule))
        hat_inputs = K.permute_dimensions(hat_inputs, (0, 2, 1, 3))

        b = K.zeros_like(hat_inputs[:, :, :, 0])
        for i in range(self.routings):
            c = softmax(b, 1)
            o = self.activation(K.batch_dot(c, hat_inputs, [2, 2]))
            if i < self.routings - 1:
                b = K.batch_dot(o, hat_inputs, [2, 3])
                if K.backend() == 'theano':
                    o = K.sum(o, axis=1)

        return o

    def compute_output_shape(self, input_shape):
        return (None, self.num_capsule, self.dim_capsule)
```

Figure 7: Capsule function

[24]:

```python
output = Conv2D(256, kernel_size=(9, 9), strides=(1, 1), activation='relu')(base_model.get_layer(name='block5_pool').output)

x = Reshape((-1, 256))(output)
capsule = Capsule(2, 16, 4, True)(x)
output = Lambda(lambda z: K.sqrt(K.sum(K.square(z), 2)))(capsule)
model = Model(inputs=input_image, outputs=output)

model.summary()
```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 299, 299, 3)	0
block1_conv1 (Conv2D)	(None, 299, 299, 64)	1792
block1_conv2 (Conv2D)	(None, 299, 299, 64)	36928
block1_pool (MaxPooling2D)	(None, 149, 149, 64)	0
block2_conv1 (Conv2D)	(None, 149, 149, 128)	73856
block2_conv2 (Conv2D)	(None, 149, 149, 128)	147584
block2_pool (MaxPooling2D)	(None, 74, 74, 128)	0
block3_conv1 (Conv2D)	(None, 74, 74, 256)	295168
block3_conv2 (Conv2D)	(None, 74, 74, 256)	590080
block3_conv3 (Conv2D)	(None, 74, 74, 256)	590080
block3_pool (MaxPooling2D)	(None, 37, 37, 256)	0
block4_conv1 (Conv2D)	(None, 37, 37, 512)	1180160
block4_conv2 (Conv2D)	(None, 37, 37, 512)	2359808
block4_conv3 (Conv2D)	(None, 37, 37, 512)	2359808
block4_pool (MaxPooling2D)	(None, 18, 18, 512)	0
block5_conv1 (Conv2D)	(None, 18, 18, 512)	2359808
block5_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block5_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block5_pool (MaxPooling2D)	(None, 9, 9, 512)	0
conv2d_1 (Conv2D)	(None, 1, 1, 256)	10617088
reshape_1 (Reshape)	(None, 1, 256)	0
capsule_1 (Capsule)	(None, 2, 16)	8192
lambda_1 (Lambda)	(None, 2)	0

```
Total params: 25,339,968
Trainable params: 17,704,704
Non-trainable params: 7,635,264
```

Figure 8: VGG-16 CapsNet summary

In [6]:

```python
import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPool2D,Dropout,Dense,Flatten
from tensorflow.keras.optimizers import Adagrad
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow.keras.preprocessing.image import load_img,array_to_img, ImageDataGenerator
from PIL import Image
from sklearn.preprocessing import OneHotEncoder
import cv2
```

In [2]:

```python
IMAGE_SIZE = [128, 128]

train_path = 'imageset/train'
valid_path = 'imageset/val'
test_path = 'imageset/test'
```

Using weights from imagenet for ResNet model

In [3]:

```python
from tensorflow.keras.applications.resnet50 import ResNet50

# add preprocessing layer to the front of VGG
vgg = ResNet50(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

# don't trains existing weights
for layer in vgg.layers:
  layer.trainable = False
```

Figure 9: importing dependencies for both models

```
[11]: #Model training and creation

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   horizontal_flip = True)

validation_datagen = ImageDataGenerator(rescale = 1./255)

test_datagen = ImageDataGenerator(rescale = 1./255)


training_set = train_datagen.flow_from_directory('imageset/train',
                                                 target_size = (128, 128),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')

validation_set = validation_datagen.flow_from_directory('imageset/val',
                                                        target_size = (128, 128),
                                                        batch_size = 32,
                                                        class_mode = 'categorical')

checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("model_vgg16.h5",
                                                   save_best_only=True)

early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience=5,
                                                     restore_best_weights=True)

def exponential_decay(lr0, s):  #
    def exponential_decay_fn(epoch):
        return lr0 * 0.1 **(epoch / s)
    return exponential_decay_fn

exponential_decay_fn = exponential_decay(0.01, 20)

lr_scheduler = tf.keras.callbacks.LearningRateScheduler(exponential_decay_fn)


# fit the model
r = model.fit_generator(
  training_set,
  validation_data=validation_set,
  epochs=5,
  steps_per_epoch=len(training_set),
  validation_steps=len(validation_set),
  callbacks=[checkpoint_cb, early_stopping_cb, lr_scheduler]
)
```

Figure 10: model summary of ResNet 50