National College of Ireland

Configuration Manual

MSc Research Project
MSc. Data Analytics

Vishal Kumar Yadav
Student ID: x19236239

School of Computing
National College of Ireland

Supervisor: Noel Cosgrave

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Vishal Kumar Yadav
Student ID:	x19236239
Programme:	MSc. Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Noel Cosgrave
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	667
Page Count:	5

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:	Vishal Kumar Yadav
Date:	23rd September 2021

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

Vishal Kumar Yadav
x19236239

1 System Configuration

This configuration manual is build to describe the research work execution and make it readily replicate for a further study or future projects. All steps have been described in this manual that were performed to execute this research work. Two deep learning techniques have been implemented on google colab pro cloud environment. Minimum hardware configuration that are required.

1. Processor: Intel Core i5 or above.
2. GPU: Intel HD Graphics 6000 1536 MB
3. RAM: memory:8 GB 1600 MHz DDR3

2 Environmental Setup

This section provides insight into the use of data mining to segment driving images in detail. While training the network, deep learning models take longer to process the pictures. The experiment is conducted on a Google Co-lab PRO cloud machine with a 100 GB hard drive, 12.72 GB RAM as well as 48.97 GB run-time GPU. Once the hardware requirements are met, necessary software packages are installed and imported.

```
!pip install git+https://github.com/qubvel/segmentation_models.pytorch
!pip install imgaug==0.2.7
!pip install --force-reinstall albumentations==1.0.3
!pip install matplotlib==3.1.3

import torch

import cv2
import matplotlib.pyplot as plt
import segmentation_models_pytorch as smp
import os
import numpy as np
from torch.utils.data import DataLoader
from torch.utils.data import Dataset as BaseDataset
```

Listing 1: Necessary Libraries

PyTorch segmentation models are used to build the deep learning model. For performing augmentation imgaug library is installed. Albumentation library is install to perfrom augumentation on the dataset by targeting the data on various aspects. Matplotlib library is used for visualisation of the data and output. Utils.data is used for data loader to flexibly use the image dataset.

3 Data Modeling

Data selected for this research work is Camvid (Cambridge-driving Labeled Video Database) dataset. CamVid dataset is publicly available. Here are the links from where dataset can be downloaded and used for future study.

1. http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/
2. mi.eng.cam.ac.uk/research/projects/VideoRec/CamSeq01/
3. kaggle datasets download -d carlolepelaars/camvid

3.1 Loading and reading the Dataset

initially the data is uploaded on the google cloab notebook and combined to read using python package 'OS' and thereby using data given in camvid dataset. Thus import os and os.path.join function are used to for accessing the train test and validation data stored in directory for further analysis.

```
x_train = os.path.join(DATA_DIR, 'train')
y_train = os.path.join(DATA_DIR, 'trainannot')

x_valid = os.path.join(DATA_DIR, 'val')
y_valid = os.path.join(DATA_DIR, 'valannot')

x_test = os.path.join(DATA_DIR, 'test')
y_test = os.path.join(DATA_DIR, 'testannot')
```

Figure 1: data loading

```
class Dataset(BaseDataset):


    CLASSES = ['sky', 'building', 'pole', 'road', 'pavement',
               'tree', 'signsymbol', 'fence', 'car',
               'pedestrian', 'bicyclist', 'unlabelled']

    def __init__(
            self,
            images_dir,
            masks_dir,
            classes=None,
            augmentation=None,
            preprocessing=None,
    ):
        self.ids = os.listdir(images_dir)
        self.images_fps = [os.path.join(images_dir, image_id) for image_id in self.ids]
        self.masks_fps = [os.path.join(masks_dir, image_id) for image_id in self.ids]


        self.class_values = [self.CLASSES.index(cls.lower()) for cls in classes]
```

Figure 2: data loading

3.2 Defining pre-processing Transformation Parameters

Once the data is ready. It is called and class dataset is defined with various measures such as setting path to image directory, path to segmentation mask folder, values and ids of class and images to extract from segmentation mask. Data transformation

pipeline is defined using albmentation. Compose function for performing various necessary transformation of images such as horizontal flip, padding, scale rotation etc. Then pre-processing function is defined for resizing and normalisation of the data.

```
    def __getitem__(self, i):


        image = cv2.imread(self.images_fps[i])
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        mask = cv2.imread(self.masks_fps[i], 0)

        masks = [(mask == v) for v in self.class_values]
        mask = np.stack(masks, axis=-1).astype('float')


        if self.augmentation:
            sample = self.augmentation(image=image, mask=mask)
            image, mask = sample['image'], sample['mask']


        if self.preprocessing:
            sample = self.preprocessing(image=image, mask=mask)
            image, mask = sample['image'], sample['mask']

        return image, mask

    def __len__(self):
        return len(self.ids)
```

Figure 3: data preprocessing

Once necessary functions has been defined, data visualisation is performed on data set to check the segmented mask associated with class images.

```
def get_training_augmentation():
    train_transform = [

        alb.HorizontalFlip(p=0.5),

        alb.ShiftScaleRotate(scale_limit=0.5, rotate_limit=0, shift_limit=0.1, p=1, border_mode=0),

        alb.PadIfNeeded(min_height=320, min_width=320, always_apply=True, border_mode=0),
        alb.RandomCrop(height=320, width=320, always_apply=True),

        alb.IAAAdditiveGaussianNoise(p=0.2),
        alb.IAAPerspective(p=0.5),

        alb.OneOf(
            [
                alb.CLAHE(p=1),
                alb.RandomBrightness(p=1),
                alb.RandomGamma(p=1),
```

Figure 4: data augumentation

Since image data is small, data loaded is augmented and transformed using albumentation library for training and testing purpose.

```python
def get_validation_augmentation():

    test_transform = [
        alb.PadIfNeeded(384, 480)
    ]
    return alb.Compose(test_transform)


def to_tensor(x, **kwargs):
    return x.transpose(2, 0, 1).astype('float32')


def get_preprocessing(preprocessing_fn):


    _transform = [
        albu.Lambda(image=preprocessing_fn),
        albu.Lambda(image=to_tensor, mask=to_tensor),
    ]
    return albu.Compose(_transform)
```

Figure 5: data loading

3.3 Data modeling for training and testing

after pre-processing and transformation of data. The augmented data is used for model training and testing purpose. Here is segmentation model based on U-Net architecture is modelled. That will process segmentation on camvid dataset and provide segmented mask results based on classes.

```python
ENCODER = 'resnet50'
ENCODER_WEIGHTS = 'imagenet'
CLASSES = ['car']
ACTIVATION = 'sigmoid' #'softmax' for multicalss segmentation



model = smp.Unet(
    encoder_name=ENCODER,
    encoder_weights=ENCODER_WEIGHTS,
    classes=len(CLASSES),
    activation=ACTIVATION,
)

preprocessing_fn = smp.encoders.get_preprocessing_fn(ENCODER, ENCODER_WEIGHTS)
```

```
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to /root/
100% 97.8M/97.8M [00:00<00:00, 206MB/s]
```

Figure 6: Defining Parameters

Once encoder and encoder weights are initialised in the architecture model is trained and tested on the data set. The Evaluation parameters are also defined in order to check the model performance on segmentation task.

Once model is tested, the results are analysed and visualised based on evaluation metrics and segmentation mask results based on random selection of images and results of test data are compared with segmentation mask generated for particular class from training the model.

```python
for i in range(0, 20):

    print('\nEpoch: {}'.format(i))
    train_logs = train_epoch.run(train_loader)
    valid_logs = valid_epoch.run(valid_loader)

    x_epoch_data.append(i)
    train_dice_loss.append(train_logs['dice_loss'])
    train_iou_score.append(train_logs['iou_score'])
    valid_dice_loss.append(valid_logs['dice_loss'])
    valid_iou_score.append(valid_logs['iou_score'])


    if max_score < valid_logs['iou_score']:
        max_score = valid_logs['iou_score']
        torch.save(model, './best_model.pth')
        print('Model saved!')

    if i == 25:
        optimizer.param_groups[0]['lr'] = 1e-5
        print('Decrease decoder learning rate to 1e-5!')
```

```
Epoch: 0
train: 100%|██████████| 46/46 [06:16<00:00,  8.18s/it, dice_loss - 0.118, iou_score - 0.8375]
valid: 100%|██████████| 101/101 [00:59<00:00,  1.70it/s, dice_loss - 0.3306, iou_score - 0.7188]
Model saved!
```

Figure 7: training

```python
# evaluate model on test set
test_epoch = smp.utils.train.ValidEpoch(
    model=best_model,
    loss=loss,
    metrics=metrics,
    device=DEVICE,
)

logs = test_epoch.run(test_dataloader)
```

```
valid: 100%|██████████| 233/233 [00:07<00:00, 32.03it/s, dice_loss - 0.2835, iou_score - 0.7518]
```

Figure 8: Testing phase

References