Configuration Manual

MSc Research Project
Data Analytics

Aman Khanna
Student ID: x19231938

School of Computing
National College of Ireland

Supervisor: Dr. Bharathi Chakravarthi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aman Khanna
Student ID:	x19231938
Programme:	Data Analytics
Year:	2020-2021
Module:	MSc Research Project
Supervisor:	Dr. Bharathi Chakravarthi
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	555
Page Count:	7

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:	22nd 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

Aman Khanna
x19231938

1 Hardware/Software Requirements

The configuration manual outlines the procedures that must be followed when executing the scripts used in the research study. This documentation will assist you in successfully running the code. This documentation also contains details on the hardware configuration of the system on which the code was run. The system's minimal necessary configuration is also given.

2 System Specification

2.1 Hardware Requirements:

The following are the hardware specifications for the system on which the research project is implemented.
Processor: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GH
RAM: 8GB
Storage: 256 SSD + 1TB HDD
Operating System: Windows 10 Home, 64-bit operating system.

2.2 Software Requirements:

This research project used following programming tools:

- Google Colaboratory (Cloud based Jupyter Notebook Environment provided by Google)
- Python Version 3
- Overleaf

3 Environment Setup

This section will help to understand the Google Colab environment. Below screenshot is included to help and guide to replicate the research.

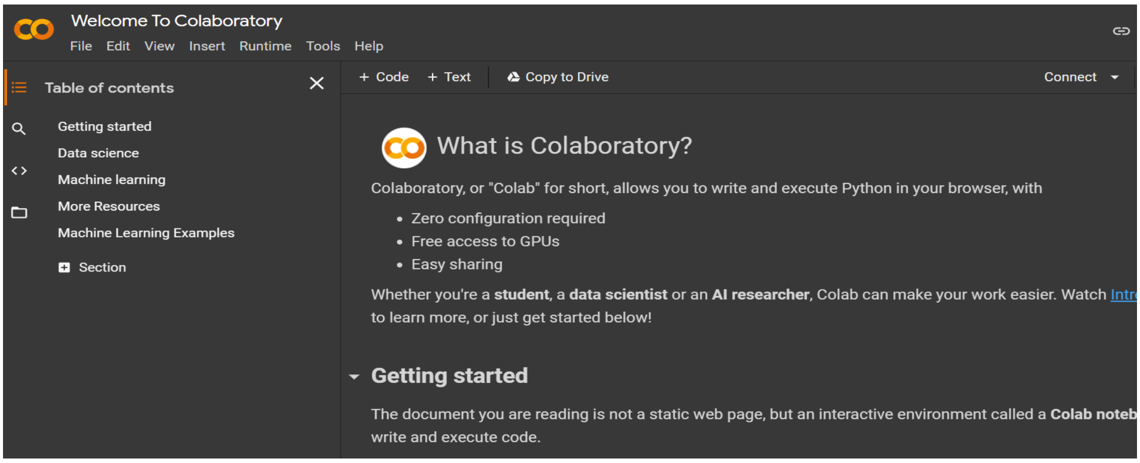

Figure 1: Google Colab

4 Data Source

This research project utilized the X-ray Images dataset(Kermany et al. (2018)) obtained from mendeley data where datas is publicly available to use as it is open source.

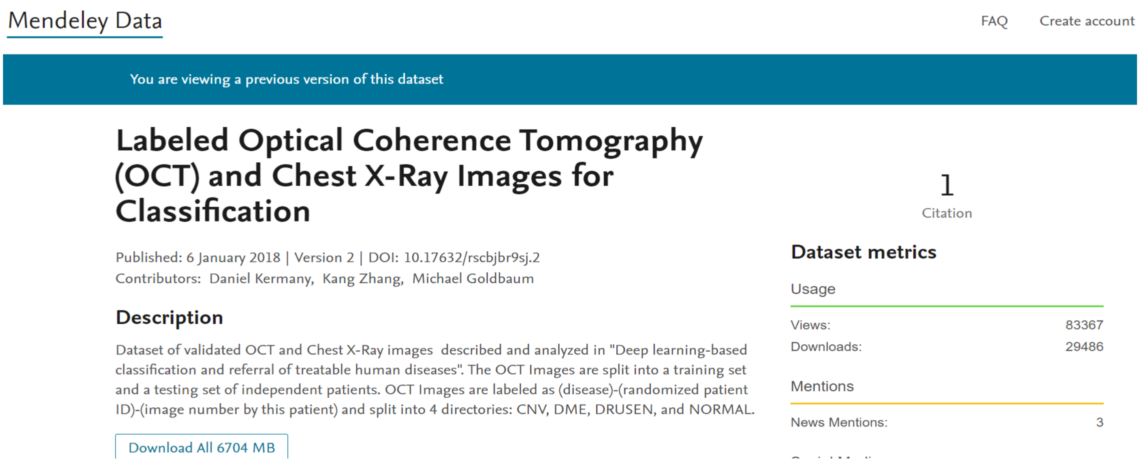

Figure 2: Dataset

Data was downloaded and uploaded on the google drive in a zip format. Further, the data was unzipped on Google drive and accessed there to prevent the re-uploading of data every time the code is run.

5 Implementation

The following libraries are used to build and perform this research.
1. Tensorflow
2. Keras
3. Numpy
4. Pandas
5. Keras-preprocessing

6. Kerastuner
7. Efficient Net
8. VGG
9. Sklearn

5.1 Data Preprocessing:

As part of data preprocessing, data augmentation was performed to carry out the zooming, flipping and rotating for the training dataset to balance out the data. Below figure 3 shows the function for the data augmentation.

Data Augmentation

```
[ ]  train_augmentation = image.ImageDataGenerator(
                    rescale=1./255,
                    rotation_range=30,
                    width_shift_range=0.1,
                    height_shift_range=0.1,
                    zoom_range=0.2,
                    horizontal_flip=True)
```

```
[ ]  test_augmentation = image.ImageDataGenerator(rescale=1./255)
```

```
[ ]  val_augmentation = image.ImageDataGenerator(rescale=1./255)
```

Figure 3: Data Augmentation

Once preprocessing was done, below function was used to examine the images.

```
[ ]  i=0
     for batch in train_generator:
         plt.figure(figsize=(12,8))
         for j in range(8):
             plt.subplot(2,4,j+1)
             imgplot = plt.imshow(image.array_to_img(batch[0][j]),cmap = 'gray')
             plt.title(batch[1][j])

         i+=1
         if i != 0:
             break

     plt.show()
```

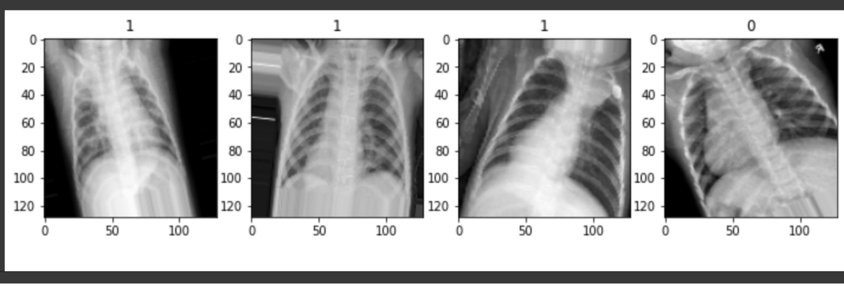

Figure 4: Preprocessed Images

5.2 Models:

5.2.1 VGG:

Below figure 5 shows the developing utilization of pre trained weights for VGG and further added more layers to it for the fine tuning. Once the model was run, more fine tuning was done to enhance the performance of the model.

```
[ ] from keras.applications.vgg16 import VGG16

[ ] image_size = 256

[ ] vgg_model = VGG16(input_shape= (image_size, image_size,3),
                      weights='imagenet',
                      include_top=False)

    vgg_model.trainable = False

[ ] transfer_model_vgg = Sequential()
    transfer_model_vgg.add(vgg_model)
    transfer_model_vgg.add(Flatten())
    transfer_model_vgg.add(Dense(512,activation='relu'))
    transfer_model_vgg.add(Dense(128,activation='relu'))
    transfer_model_vgg.add(Dense(1,activation='sigmoid'))

[ ] transfer_model_vgg.summary()
```

Figure 5: VGG Net

5.2.2 Efficient Net:

Below figure 6 shows the developing utilization of Efficient Net with pre-trained wieghts of Imagenet and making some fine tuning changes with the model to increase the performance.

5.2.3 CNN- Kerastuner:

Below figures 7, 8 & 9 shoes the formation of CNN model and making of model search function and further to search the best parameters for the model using tuner search method.

Now, once all the models were ready, evaluation of the model was performed to get the best model among them using different metrices.

6 Other Software:

The documentation of the research was carried out using the Overleaf. Below figure 10 depicts how the overleaf was used for the project.

```python
## Keep the best model
mc = ModelCheckpoint('model.hdf5',
                     save_best_only=True,
                     verbose=0,
                     monitor='val_loss',
                     mode='min')

## Reduce learning rate if it gets stuck in a plateau
rlr = ReduceLROnPlateau(monitor='val_loss',
                        factor=0.3,
                        patience=3,
                        min_lr=0.000001,
                        verbose=1)

# Model
## Define the base model with EfficientNet weights
model = efn.EfficientNetB4(weights = 'imagenet',
                           include_top = False,
                           input_shape = (SIZE, SIZE, 3))

## Output layer
x = model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation="relu")(x)
x = Dense(32, activation="relu")(x)
predictions = Dense(1, activation="sigmoid")(x)

## Compile and run
model = Model(inputs=model.input, outputs=predictions)

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy', Recall(), Precision()])
```

Figure 6: Efficient Net

```python
[ ]  model=Sequential([
             Input(shape=(128,128,3)),
             Conv2D(32,3,activation='relu'),
             MaxPool2D(pool_size=2),
             Conv2D(32,3,activation='relu'),
             MaxPool2D(pool_size=3),
             Flatten(),
             Dense(100,activation='relu'),
             Dense(1,activation='sigmoid')
     ])
```

Figure 7: CNN model

```
[ ] import kerastuner

[ ] def model_builder(hp):
      model = keras.Sequential()
      model.add(keras.layers.Flatten(input_shape=(128, 128,3)))

      # Tune the number of units in the first Dense layer
      # To choose an optimal value between 32-512
      hp_units = hp.Int('units', min_value=32, max_value=512, step=32)
      model.add(keras.layers.Dense(units=hp_units, activation='relu'))
      model.add(keras.layers.Dense(10))

      # Tune the learning rate for the optimizer
      # Choose an optimal value from 0.01, 0.001, or 0.0001
      hp_learning_rate = hp.Choice('learning_rate', values=[0.01, 0.001, 0.0001])

      model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                    metrics=['accuracy'])

      return model

[ ] tuner = kerastuner.Hyperband(model_builder,
                         objective='val_accuracy',
                         max_epochs=10,
                         factor=3,
                         directory='my_dir',
                         project_name='intro_to_kt')
```

Figure 8: Function for the model search using kerastuner

```
[ ] tuner.search(train_generator,validation_data=valid_generator, epochs=50,batch_size=32, callbacks=[stop_early])

    # Get the optimal hyperparameters
    best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

    print(f"""
    The hyperparameter search is complete. The optimal number of units in the first densely-connected
    layer is {best_hps.get('units')} and the optimal learning rate for the optimizer
    is {best_hps.get('learning_rate')}.
    """)

    Trial 30 Complete [00h 10m 48s]
    val_accuracy: 0.875

    Best val_accuracy So Far: 0.9375
    Total elapsed time: 02h 13m 09s
    INFO:tensorflow:Oracle triggered exit

    The hyperparameter search is complete. The optimal number of units in the first densely-connected
    layer is 256 and the optimal learning rate for the optimizer
    is 0.01.

[ ] # Building the model with the optimal hyperparameters and training it on the data for 50 epochs
    model = tuner.hypermodel.build(best_hps)
    history = model.fit(train_generator,validation_data=valid_generator, epochs=30,batch_size=32)

    val_acc_per_epoch = history.history['val_accuracy']
    best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
    print('Best epoch: %d' % (best_epoch,))
```

Figure 9: Getting the hyper paramters for the model with tuner search functions of kerastuner

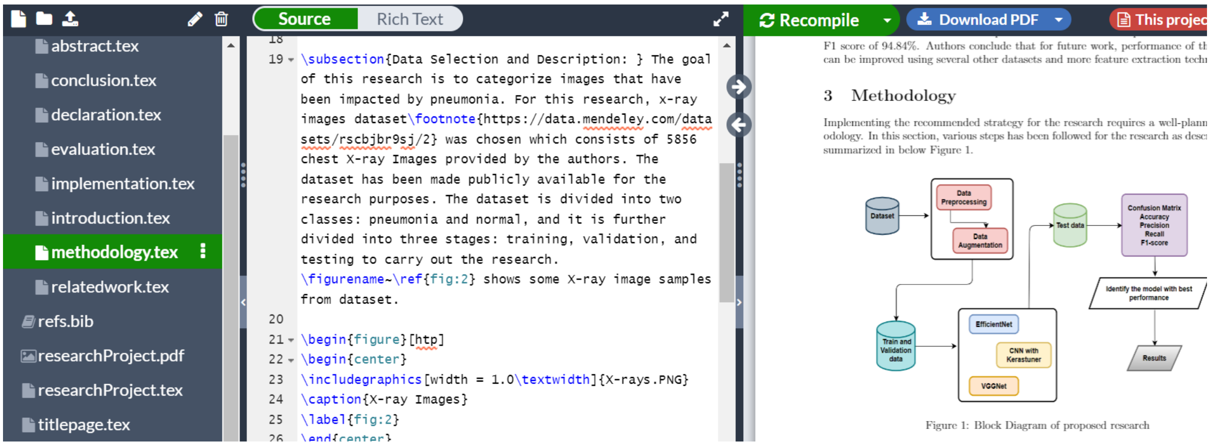

Figure 10: Overleaf Environment

References

Kermany, D., Zhang, K. and Goldbaum, M. (2018). Labeled optical coherence tomography (oct) and chest x-ray images for classification, *mendeley* .
URL: *https://data.mendeley.com/datasets/rscbjbr9sj/2*