Configuration Manual

MSc Research Project
MSc Data Analytics

Vishnu Venugopal Palakkath
Student ID: 20177470

School of Computing
National College of Ireland

Supervisor: Dr.Bharathi Chakravarthi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vishnu Venugopal Palakkath
Student ID:	20177470
Programme:	MSc Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr.Bharathi Chakravarthi
Submission Due Date:	16/12/2021
Project Title:	Few Shot Learning Approach to Online Malayalam Handwritten Character recognition
Word Count:	1162
Page Count:	10

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:	Vishnu Venugopal Palakkath
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

Vishnu Venugopal Palakkath
20177470

1 Introduction

The documentation for this research project includes detailed instructions on how to execute the scripts that were developed for this project's purposes. Making sure that the code runs smoothly and without errors is made significantly easy in this way. Also included is information regarding the hardware configuration of the system on which the scripts were developed, as well as the same minimal configuration that is recommended. This procedure should be followed in order to verify that the project's findings can be repeated,following which, further investigation can be carried out with relative simplicity.

2 System Configuration

2.1 Software

All throughout research, we wrote code in the Python 3.9 programming language. The Jupyter Notebook on the Anaconda Navigator platform was used to code in Python for this research.For the training of models,which we will cover in the subsequent sections,it is best to upload the notebooks on Google Colab for easy GPU access.

2.2 Hardware

The hardware requirements are as mentioned in Figure 1

Figure 1: Hardware specs for the local system

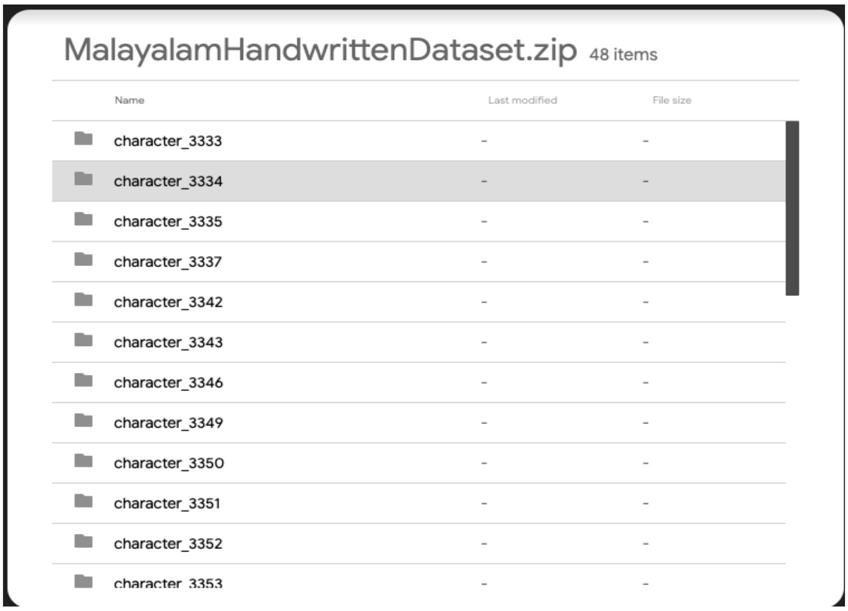

Figure 2: Malayalam Handwritten Dataset from the link

3 Python Library requirements

If the files are being opened on a local system then it is advisable to create a new python environment and run the following command.

- pip install -r requirements.txt

4 Dataset Description

- The data set is a freely available dataset at *Malayalamhandwrittendataset.zip* (n.d.).It consists of 48 characters in Malayalam with an average of 127 handwritten images for each class
- The data set is also available in the artefacts file and goes by the name Malayalam-handwrittendataset.zip.

4.1 Data Preparation

- Extract the dataset and rename it as "raw_data"
- Open the "Data Preparation-Thesis.ipynb"
- There are mainly three different functions in this notebook,Augmentation,Cropping and resizing and dataset preparation.Run all the cells one after the other to get the data.pickle file.
- There also exists a data.pickle.zip file in the artefacts folder.The dataset can be extracted directly from this for bypassing the above step.
- Navigate to the folder named Few Shot in the artefacts folder in order to Prepare data for Few Shot Learning using the Few-shot learning Data Preparation.ipynb file.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(32, 32, 1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
#model.add(Dense(64, activation='relu'))
#model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss="categorical_crossentropy",
              optimizer="adam",
              metrics=['accuracy'])
model.summary()
```

Figure 3: Code for CNN model in thesis_1.ipynb

5 Model Training

It is recommended that the notebooks mentioned in this section be run in Google colab incase of no GPU facility.

- The notebook file thesis_1.ipynb contains the training code for CNN,CNN-RNN, VGG, EfficientNet, Resnet and DenseNet.

5.1 CNN Model Training

- The CNN model is created and compiled.
- The CNN model was trained with 20 Epochs and with a batch size of 32.There are callbacks in this model for early stopping incase validation loss doesnt improve.There also callbacks to plot data using tensorboard.
- Model is evaluated on unseen data
- Accuracy of 97 percent is achieved
- Model is saved as "cnn_model.h5"

5.2 CNN-RNN Model Training

- The CNN-RNN model is created and compiled.
- The CNN-RNN model was trained with 20 Epochs and with a batch size of 32.There are callbacks in this model for early stopping incase validation loss doesnt improve.There also callbacks to plot data using tensorboard.
- Model is evaluated on unseen data
- Accuracy of 97 percent is achieved
- Model weights are saved as "CNN-RNN_weights.hdf5"

```
hog_images = []
hog_features = []
ppc = 4
for image in train_dataset:
    fd = hog(image, orientations=4, pixels_per_cell=(ppc,ppc),cells_per_block=(2, 2))
    #hog_images.append(hog_image)
    hog_features.append(fd)
```

Figure 4: HOG

```
clf1 = svm.SVC(kernel='poly')
#clf2 = naive_bayes.MultinomialNB()
clf3 = svm.SVC(kernel='rbf')
hog_features = np.array(hog_features)
data_frame = np.hstack((hog_features,labels))
#np.random.shuffle(data_frame)
```

Figure 5: SVM

5.3 HOG-SVM-POLY/RBF Model Training

- The HOG-SVM models are created and compiled.
- The HOG-SVM model was trained with 784 features.
- Model is evaluated based on Kfold cross validation.
- Accuracy of 94 percent was achieved by both models.
- Model weights are saved as “poly-svm.sav” and ”rbf-svm.sav” pickle files

5.4 VGG-16 Model Training

- The VGG-16 models are created and compiled.
- For VGG 16, only the first 9 layers are retrained out of the 16 layers.
- The VGG-16 model was trained with 20 Epochs and with a batch size of 32.There are callbacks in this model for early stopping in case validation loss doesn't improve.There also callbacks to plot data using tensorboard.
- Model is evaluated on unseen test data.
- Accuracy of 98 percent was achieved.
- Model weights are saved as “vgg_model.h5”

```python
def create_vgg16net(chexnet_weights = None):
      img_size_target = 32
      img_input = Input(shape=(img_size_target, img_size_target, 1))
      img_conc = Concatenate()([img_input, img_input, img_input])
      model =VGG16(
                    include_top=False, weights="imagenet", input_tensor=img_conc,
                    input_shape=(32,32,3), pooling=None, classes=1000,
                    classifier_activation='softmax') #importing densenet the last layer will be

      #we need to load the weights so setting the architecture of the model as same as the one
      x = model.layers[-2].output #output from vgg16net
      x = GlobalAveragePooling2D()(x)
      x = Dense(num_classes, activation="sigmoid", name="vgg16net_output")(x) #here activation

      vgg16net = tf.keras.Model(inputs = model.input,outputs = x)
      #vgg16net.load_weights(chexnet_weights)
      #vgg16net = tf.keras.Model(inputs = model.input,outputs = vgg16net.layers[-2].output)  #w
      return vgg16net
```

Figure 6: VGG

```python
model.get_layer('block5_conv1').trainable = False
model.get_layer('block5_conv2').trainable = False
model.get_layer('block5_conv3').trainable = False

model.get_layer('block4_conv1').trainable = False
model.get_layer('block4_conv2').trainable = False
model.get_layer('block4_conv3').trainable = False

model.get_layer('block3_conv1').trainable = False
model.get_layer('block3_conv2').trainable = False
model.get_layer('block3_conv3').trainable = True

model.get_layer('block2_conv1').trainable = True
model.get_layer('block2_conv2').trainable = True

model.get_layer('block1_conv1').trainable = True
model.get_layer('block1_conv2').trainable = True


model.get_layer('block1_pool').trainable = True
model.get_layer('block2_pool').trainable = True
model.get_layer('block3_pool').trainable = True
model.get_layer('block4_pool').trainable = True
model.get_layer('global_average_pooling2d').trainable = True

#model.get_layer('block2_conv3').trainable = False


model.summary()
```

Figure 7: VGG layers

Efficientnet

```
def create_effnet(chexnet_weights = None):
      img_size_target = 32
      img_input = Input(shape=(img_size_target, img_size_target, 1))
      img_conc = Concatenate()([img_input, img_input, img_input])
      model =EfficientNetB2(
                    include_top=False, weights="imagenet", input_tensor=img_conc,
                    input_shape=(32,32,3), pooling=None, classes=1000,classifier_activation='so
                    ) #importing densenet the last layer will be a relu activation layer#Effici

      #we need to load the weights so setting the architecture of the model as same as the one
      x = model.layers[-2].output #output from vgg16net
      x = GlobalAveragePooling2D()(x)
      x = Dense(num_classes, activation="softmax", name="effnet_output")(x) #here activation is

      effnet = tf.keras.Model(inputs = model.input,outputs = x)
      return effnet

effnet=create_effnet()
model = tf.keras.Model(inputs = effnet.input, outputs = effnet.output)

for i in range(0,341):#476 layers #300,7
   model.layers[i].trainable = False
for i in range(0,341,33):#476 layers #300,7
   model.layers[i].trainable = True
# for i in range(0,50):#476 layers #300,7
#    model.layers[i].trainable = True
```

Figure 8: Efficientnet

```
for i in range(0,341):#476 layers #300,7
   model.layers[i].trainable = False
for i in range(0,341,33):#476 layers #300,7
   model.layers[i].trainable = True
```

Figure 9: Efficientnet layers

5.5 EfficientNet Model Training

- The EfficientNet model is created and compiled.
- In EfficientNetB2,every 33rd layer until the layer 341 was enabled for training.
- The EfficientNet model was trained with 20 Epochs and with a batch size of 32.There are callbacks in this model for early stopping incase validation loss doesnt improve.There also callbacks to plot data using tensorboard.
- Model is evaluated on unseen test data.
- Accuracy of 84 percent was achieved .
- Model was saved as ”effnet_model.h5”

5.6 ResNet Model Training

- The ResNet50 models is created and compiled.
- Whereas in the Resnet50 ,the first 50 layers and all layers that are multiples of 33 out of 177 layers are trained.

```
def create_effnet(chexnet_weights = None):
      img_size_target = 32
      img_input = Input(shape=(img_size_target, img_size_target, 1))
      img_conc = Concatenate()([img_input, img_input, img_input])
      model =ResNet50(
                    include_top=False, weights="imagenet", input_tensor=img_conc,
                    input_shape=(32,32,3), pooling=None, classes=1000,classifier_activation='so
                    )

      #we need to load the weights so setting the architecture of the model as same as the one
      x = model.layers[-2].output #output from vgg16net
      x = GlobalAveragePooling2D()(x)
      x = Dense(num_classes, activation="softmax", name="effnet_output")(x) #here activation is

      effnet = tf.keras.Model(inputs = model.input,outputs = x)
      return effnet

effnet=create_effnet()
model = tf.keras.Model(inputs = effnet.input, outputs = effnet.output)

for i in range(0,341):#476 layers #300,7
   model.layers[i].trainable = False
for i in range(0,341,33):#476 layers #300,7
   model.layers[i].trainable = True
# for i in range(0,50):#476 layers #300,7
#    model.layers[i].trainable = True
```

Figure 10: Resnet

- The ResNet50 model was trained with 20 Epochs and with a batch size of 32.There are callbacks in this model for early stopping in case validation loss doesn't improve.There also callbacks to plot data using tensorboard.
- Model is evaluated on unseen test data.
- Accuracy of 97 percent was achieved.
- Model weights are saved as "resnet_model.h5"

5.7 DenseNet Model Training

- The DenseNet models is created and compiled.
- In DenseNet ,every 9th layer until layer 300 was enabled for training.
- The DenseNet model was trained with 20 Epochs and with a batch size of 32.There are callbacks in this model for early stopping in case validation loss doesn't improve.There also callbacks to plot data using tensorboard.
- Model is evaluated on unseen test data.
- Accuracy of 98 percent was achieved.
- Model weights are saved as "DenseNet_model.h5"

```python
from tensorflow.keras.metrics import Precision,PrecisionAtRecall,AUC,Recall,Accuracy

from sklearn.metrics import classification_report

def create_effnet(chexnet_weights = None):
      img_size_target = 32
      img_input = Input(shape=(img_size_target, img_size_target, 1))
      img_conc = Concatenate()([img_input, img_input, img_input])
      model =DenseNet(
                    include_top=False, weights="imagenet", input_tensor=img_conc,
                    input_shape=(32,32,3), pooling=None, classes=1000
                    ) #importing densenet the last layer will be a relu activation layer#Effic

      #we need to load the weights so setting the architecture of the model as same as the one
      x = model.layers[-2].output #output from vgg16net
      x = GlobalAveragePooling2D()(x)
      x = Dense(num_classes, activation="softmax", name="effnet_output")(x) #here activation is

      effnet = tf.keras.Model(inputs = model.input,outputs = x)
      return effnet

effnet=create_effnet()
model = tf.keras.Model(inputs = effnet.input, outputs = effnet.output)

for i in range(0,341):#476 layers #300,7
   model.layers[i].trainable = False
for i in range(0,300,9):#476 layers #300,7
   model.layers[i].trainable = True
# for i in range(0,50):#476 layers #300,7
#    model.layers[i].trainable = True
```

Figure 11: DenseNet

```python
for i in range(0,341):#476 layers #300,7
   model.layers[i].trainable = False
for i in range(0,300,9):#476 layers #300,7
   model.layers[i].trainable = True
# for i in range(0,50):#476 layers #300,7
#    model.layers[i].trainable = True
```

Figure 12: DenseNet layers

```python
def get_siamese_model(input_shape):
    left_input = Input(input_shape)
    right_input = Input(input_shape)
    model = Sequential()
    model.add(Conv2D(64, (3,3), activation='relu', input_shape=input_shape,
                   kernel_initializer=initialize_weights, kernel_regularizer=l2(2e-4)))
    model.add(MaxPooling2D())
    model.add(Conv2D(128, (3,3), activation='relu',
                     kernel_initializer=initialize_weights,
                     bias_initializer=initialize_bias, kernel_regularizer=l2(2e-4)))
    model.add(MaxPooling2D())
    model.add(Conv2D(128, (3,3), activation='relu', kernel_initializer=initialize_weights,
                     bias_initializer=initialize_bias, kernel_regularizer=l2(2e-4)))
    model.add(MaxPooling2D())
    model.add(Conv2D(256, (2,2), activation='relu', kernel_initializer=initialize_weights,
                     bias_initializer=initialize_bias, kernel_regularizer=l2(2e-4)))
    model.add(GlobalMaxPooling2D())
    model.add(Flatten())
    model.add(Dense(4096, activation='sigmoid',
                   kernel_regularizer=l2(1e-3),
                   kernel_initializer=initialize_weights,bias_initializer=initialize_bias))
    encoded_l = model(left_input)
    encoded_r = model(right_input)
    L1_layer = Lambda(lambda tensors:K.abs(tensors[0] - tensors[1]))
    L1_distance = L1_layer([encoded_l, encoded_r])
    prediction = Dense(1,activation='sigmoid',bias_initializer=initialize_bias)(L1_distance)
    siamese_net = Model(inputs=[left_input,right_input],outputs=prediction)
    return siamese_net
```

Figure 13: Siamese Network

5.8 SiameseNet-CNN Training

- In order to train the siamese Networks,navigate to the Few Shot folder from artefacts and open the Few_shot_learning.ipynb file.
- The Siamese Net models are created and compiled.
- The Siamese Net model was trained for 100 epochs with a batch size of 1000,20 classes per subset and 550 validations.
- Model is evaluated based 20 way 550 one-shot classification tasks.
- Accuracy of 100 percent on training and testing data percent was achieved by both models.
- Model weights are saved as "OneShot_weights.h5" and "OneShot_weights_LSTM.h5" files

6 Model Prediction

- In order to make predictions for the characters go to the file "Thesis Prediction.ipynb"
- Go to the Cell below the heading Predictions for ML/DL Models
- When that cell is run it shows a figure that looks like figure 8

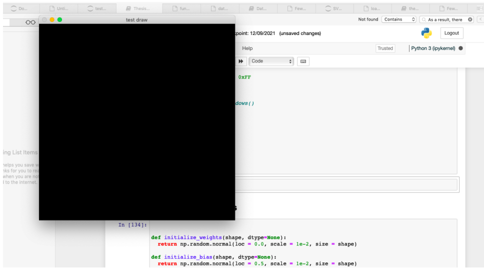

Figure 14: blackscreen

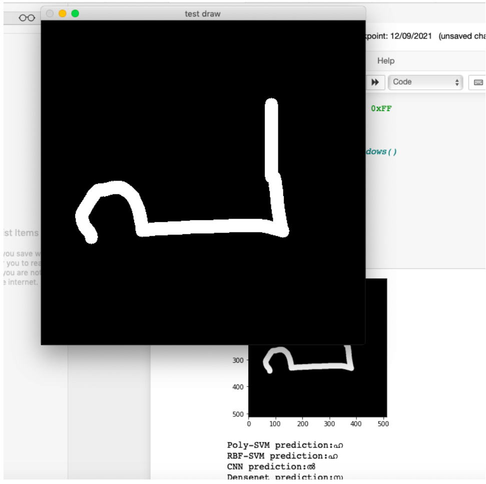

Figure 15: Write on the black screen

- As you scribble on the trackpad,it would look like Figure 9
- The screen is reset as you finish one writing on the trackpad
- As the screen is reset ,the predictions of each model start coming in
- Make sure to write continuously because if you put breaks in between ,it will be considered as 1 character.
- To try out Few shot prediction navigate to the last cell in the notebook and repeat steps 2-6

References

Malayalamhandwrittendataset.zip (n.d.).
URL: *https://drive.google.com/file/d/1WjZnnmmfjv7-N-WakhJdLoDhiHEi5dOb/view*