National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Palak
Student ID: 18185461

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Palak		
Student ID:	18185461		
Programme:	MSc in Data Analytics	**Year:**	2019-2020
Module:	MSc Research Project		
Supervisor:	Dr. Vladimir Milosavljevic		
Submission Due Date:	28th September 2020		
Project Title:	Creation of Mnemonics for Hindi alphabets using CNN and Autoencoders		
Word Count:	604 (Including references) **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:	25th September 2020

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

Palak
Student ID: 18185461

1 Introduction

This document explains various aspects of the research. It elaborates on how to implement the research and overcome any possible challenges that might arise.

2 System Configuration

There are several specifications and intricacies involved with the research implementation.

2.1 Hardware Requirements

The research was implemented with the following hardware setup: -

- Processor: Intel Core i5-9300H 9th Gen processor, Quad-Core
- Operating System: Windows 10, 64Bit
- CPU: 8GB DDR4 RAM, 2666Mhz
- GPU: Nvidia GeForce GTX 1660 Ti, 6GB
- Storage: 1TB HDD + 256GB NVMe M.2 SSD

2.2 Software Requirements

The following software are required for the research: -

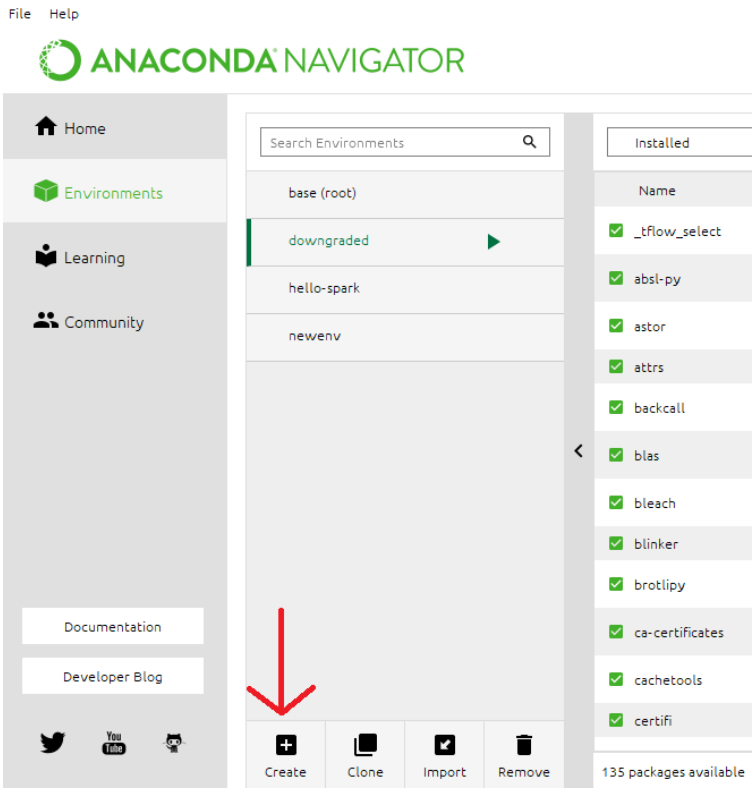
- Anaconda Navigator 1.9.7 for Windows
- Anaconda Prompt
- Jupyter Notebook 6.0.3
- Python 3.6.0
- CUDA 10.0.130
- cudnn 7.6.5

3 Research Specifications

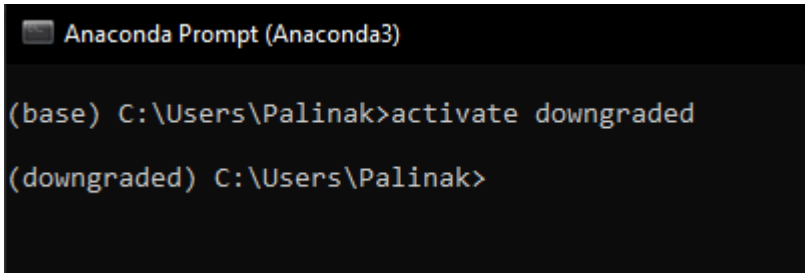
3.1 Environment Setup

The research made use of some downgraded libraries of Python to overcome incompatibility issues. Therefore, a new environment in Anaconda was set up to satisfy all the requirements. The illustration of the same is as follows: -

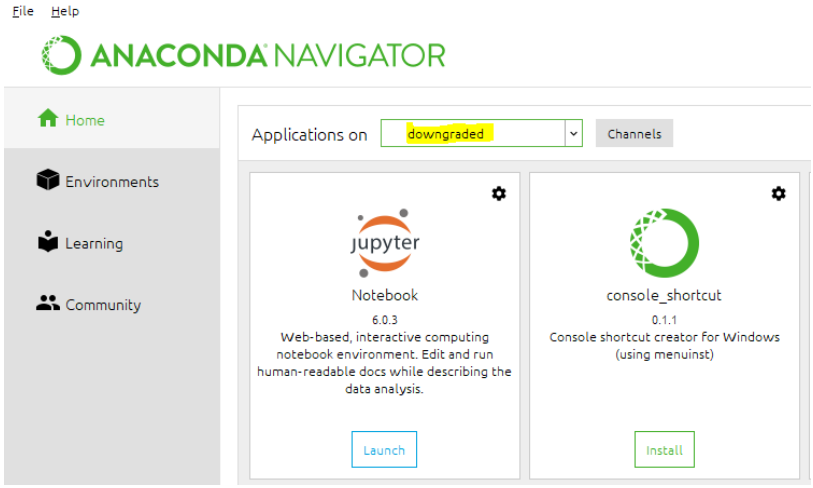
1. Open Anaconda Navigator. Click on ‘Environments’ tab. Click on ‘Create’ in the bottom.



2. Open Anaconda Prompt. Type ‘activate *your_env_name*.



3. Install all the required libraries in this environment using ‘pip install’. Before opening Jupyter notebook from Anaconda navigator, select the environment created.



3.2 Data Gathering

There were 3 datasets used in the research. All the datasets had images in png format.

1. Dataset 1: Hindi/Devanagari Font characters
   This dataset could be obtained by converting the publicly downloadable fonts into png character set. This could be done by the following code (ccs96307, 2020): -

```
import os
import re
import shutil
from PIL import Image, ImageDraw, ImageFont

def font2png(W, H, font_name, font_size, bg_color, font_color, word):
    # Font
    font = ImageFont.truetype('fonts/{}'.format(font_name), font_size)
    # Image
    image = Image.new('RGBA', (W, H), (0, 0, 0, 0))
    draw = ImageDraw.Draw(image)
    offset_w, offset_h = font.getoffset(word)
    w, h = draw.textsize(word, font=font)
    pos = ((W - w - offset_w) / 2, (H - h - offset_h) / 2)
    # Draw
    draw.text(pos, word, font_color, font=font)
    # Save png file
    for item in image.getdata():
        if item != (0, 0, 0):
            if len(os.listdir('images/{}/'.format(word))) >= 1:
                max_num = max([int(re.sub('\.png', '', png)) for png in os.listdir('images/{}/'.format(word))])
                max_num += 1
            else:
                max_num = 1

            print(word, max_num)
            image.save('images/{}/{}.png'.format(word, max_num))
            break
def main():
    # Check the 'fonts' existed
    if 'fonts' not in os.listdir('./'):
        print('Error! You have no "fonts/" folder.')
        exit()
    # Settings
    words = open('inputs.txt',encoding="utf-8").read().split('\n')
    W, H = (256, 256)
    font_size = 256
    background_color = 'white'
    font_color = 'black'
    font_path = 'fonts/'
    # Remove existed folder
    if 'images' in os.listdir('./'):
        shutil.rmtree('images')
    os.mkdir('images/')
    # Save the font images in echo other folder
    for word in words:
        try:
            os.mkdir('images/{}'.format(word))
        except:
            pass
        for font_name in os.listdir(font_path):
            try:
                font2png(W, H, font_name, font_size, background_color, font_color, word)
            except:
                continue
if __name__ == '__main__':
    main()
```

2. Dataset 2: Hindi/Devanagari handwritten characters
   It is publicly available on Kaggle (Jha, 2018).

3. Dataset 3: Potential Mnemonic images
   This dataset is flexible and can be changed according to anyone's wish. These can be any images ranging from chairs and tables to taps and birds. It should, however, be kept in mind that the data should be unrestricted and publicly available with all the required grants and permissions.

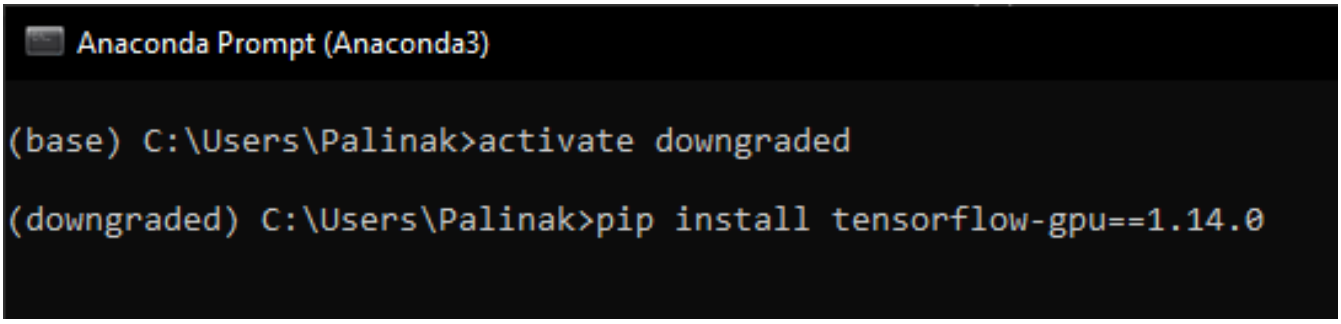
   This research utilized datasets from a few dataset from Kaggle (Roy, Bhattacharya and Ghosh, 2018) (Zhang, 2019).

3.3 Libraries required

The libraries and their versions required for the research are: -

- tensorflow-gpu 2.1.6
- keras-gpu 1.14.0
- numpy 1.18.5
- matplotlib 3.2.2
- open-cv (cv2) 3.3.1
- pil 7.1.2
- scipy 1.5.0
- skimage 0.17.2
- os
- pickle
- shutil
- re

Install the libraries by opening Anaconda prompt and typing 'pip install *library name*



3.4 Research Code

The entire code would be submitted as artefacts. The important snippets of the code to be implemented for various stages of the research are as follows: -

3.4.1 Implementing CNN for Hindi handwriting recognition

1. Create training data

```
import numpy as np
import os
from matplotlib import pyplot as plt
import cv2
import random
import pickle


file_list = []
class_list = []

DATADIR = "D:/hindi/DevanagariHandwrittenCharacterDataset/Train/"

# All the categories you want your neural network to detect
CATEGORIES = ["character_1_ka", "character_2_kha", "character_3_ga", "character_4_gha", "character_5_kna", "character_6_cha",
              "character_7_chha", "character_8_ja", "character_9_jha", "character_10_yna", "character_11_ta", "character_12_thaa
              "character_13_daa", "character_14_dhaa", "character_15_adna", "character_16_tabala", "character_17_tha", "characte
              "character_19_dha", "character_20_na", "character_21_pa", "character_22_pha", "character_23_ba", "character_24_bha
              "character_25_ma","character_26_yaw", "character_27_ra", "character_28_la", "character_29_waw", "character_30_moto
              "character_31_petchiryakha", "character_32_patalosaw", "character_33_ha", "character_34_chhya", "character_35_tra"
              "digit_0", "digit_1", "digit_2", "digit_3", "digit_4", "digit_5", "digit_6", "digit_7", "digit_8", "digit_9"]

# The size of the images that your neural network will use
IMG_SIZE = 32

# Checking or all images in the data folder
for category in CATEGORIES :
    path = os.path.join(DATADIR, category)
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)

training_data = []

def create_training_data():
    for category in CATEGORIES :
        path = os.path.join(DATADIR, category)
        class_num = CATEGORIES.index(category)
        for img in os.listdir(path):
            try :
                img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                training_data.append([new_array, class_num])
            except Exception as e:
                pass

create_training_data()
```

2. Building and training the model

```
# Building the model
model = Sequential()
# 3 convolutional layers
model.add(Conv2D(32, (3, 3), input_shape = X.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# 2 hidden layers
model.add(Flatten())
model.add(Dense(128))
model.add(Activation("relu"))

model.add(Dense(128))
model.add(Activation("relu"))

# The output layer with 46 neurons, for 46 classes
model.add(Dense(46))
model.add(Activation("softmax"))

# Compiling the model using some basic parameters
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])

# Training the model, with 40 iterations
# validation_split corresponds to the percentage of images used for the validation phase compared to all the images
history = model.fit(X, y, batch_size=25, epochs=40, validation_split=0.1)

# Saving the model
model_json = model.to_json()
with open("model.json", "w") as json_file :
    json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")
model.save('CNN1.model')
```

3. Testing and Predicting

```
import cv2
import tensorflow as tf
from PIL import Image
from IPython.display import Image, display

CATEGORIES = ["character_1_ka", "character_2_kha", "character_3_ga", "character_4_gha", "character_5_kna", "character_6_cha",
              "character_7_chha", "character_8_ja", "character_9_jha", "character_10_yna", "character_11_ta", "character_12_thaa
              "character_13_daa", "character_14_dhaa", "character_15_adna", "character_16_tabala", "character_17_tha", "characte
              "character_19_dha", "character_20_na", "character_21_pa", "character_22_pha", "character_23_ba", "character_24_bha
              "character_25_ma","character_26_yaw", "character_27_ra", "character_28_la", "character_29_waw", "character_30_moto
              "character_31_petchiryakha", "character_32_patalosaw", "character_33_ha", "character_34_chhya", "character_35_tra"
              "digit_0", "digit_1", "digit_2", "digit_3", "digit_4", "digit_5", "digit_6", "digit_7", "digit_8", "digit_9"]

def prepare(file):
    IMG_SIZE = 32
    img_array = cv2.imread(file, cv2.IMREAD_GRAYSCALE)
    new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
    return new_array.reshape(-1, IMG_SIZE, IMG_SIZE, 1)

model = tf.keras.models.load_model("CNN1.model")
image = prepare("D:/hindi/DevanagariHandwrittenCharacterDataset/Test/character_1_ka/1452.png")#your image path
prediction = model.predict([image])
prediction = list(prediction[0])
display(Image(filename="D:/images/"+CATEGORIES[prediction.index(max(prediction))]+".png"))
hand_char = Image(filename="D:/images/"+CATEGORIES[prediction.index(max(prediction))]+".png")
```

3.4.2 Implementing Autoencoder I (Hindi handwritten characters) and Autoencoder II (Mnemonic images)

1. Importing libraries. Preparing the data.

```
import sys
import os
from pathlib import Path
import tensorflow as tf
import keras, keras.layers as Layers, keras.backend as K
import numpy as np
from sklearn.model_selection import train_test_split
from skimage import transform
from scipy import misc
%matplotlib inline
import matplotlib.pyplot as plt
import imageio
from keras.engine import InputLayer
```

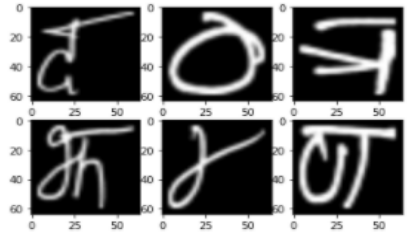
```
dataset = []
addresses = []
for img_address in Path("D:/hindi2/DevanagariHandwrittenCharacterDataset/").glob("**/*.png"):
    addresses.append(img_address)
    img = imageio.imread(img_address, pilmode="RGBA")
    #img[:,:,0:3][img[:,:,3]==0] = 255
    img = img[:,:,0:3] / 255 - 0.5 #normalizing
    img = transform.resize(img, (64, 64))
    #img = (img - np.mean(img))/np.std(img) #brigning the mean around 0.0  and the standard deviation around 1.0
    dataset.append(img)
dataset = np.array(dataset)
print(dataset.shape)
```

```
(2300, 64, 64, 3)
```

2. Exploring and visualizing the data

```
IMG_SIZE = dataset[0].shape
print("Image shape: {}".format(IMG_SIZE))
print("Image mean: {}".format(np.round(np.mean(dataset[0]))))
print("Image std: {}".format(np.std(dataset[0])))
for i in range(6):
    idx = np.random.randint(0,high=(len(dataset)),size=1)[0]
    print("{}:{}".format(i,addresses[idx]))
    plt.subplot(2,3,i+1)
    show_image(dataset[idx][:,:,:])
```

```
Image shape: (64, 64, 3)
Image mean: -0.0
Image std: 0.29648181305444865
0:D:\hindi2\DevanagariHandwrittenCharacterDataset\Test\character_18_da\1106.png
1:D:\hindi2\DevanagariHandwrittenCharacterDataset\Train\digit_0\6781.png
2:D:\hindi2\DevanagariHandwrittenCharacterDataset\Train\character_35_tra\6677.png
3:D:\hindi2\DevanagariHandwrittenCharacterDataset\Train\character_9_jha\4456.png
4:D:\hindi2\DevanagariHandwrittenCharacterDataset\Test\digit_4\5398.png
5:D:\hindi2\DevanagariHandwrittenCharacterDataset\Train\character_15_adna\6188.png
```



3. Defining the structure of Convolutional Autoencoder

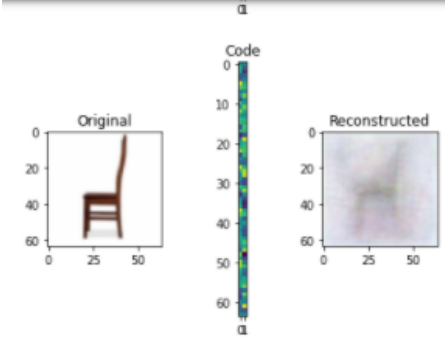
```
def covolutional_autoencoder(img_shape, code_size):

    # encoder where we compress the image information into the code
    encoder = keras.models.Sequential()
    encoder.add(Layers.InputLayer(img_shape))
    encoder.add(Layers.Conv2D(64,(3,3),activation='elu', padding='same', input_shape=img_shape))
    encoder.add(Layers.Dropout(0.2))
    encoder.add(Layers.Conv2D(128,(3,3),activation='elu', padding='same'))
    encoder.add(Layers.Dropout(0.2))
    encoder.add(Layers.Flatten())
    encoder.add(Layers.Dense(64))
    encoder.add(Layers.Dropout(0.2))
    encoder.add(Layers.Dense(code_size))

    # decoder where we expand the code into the image
    decoder = keras.models.Sequential()
    decoder.add(Layers.InputLayer((code_size,)))
    decoder.add(Layers.Dense(code_size))
    decoder.add(Layers.Dense(64*64*128))
    encoder.add(Layers.Dropout(0.2))
    decoder.add(Layers.Reshape(target_shape=(64,64,128)))
    decoder.add(Layers.Conv2DTranspose(filters=128, kernel_size=(3, 3), activation='elu', padding='same'))
    encoder.add(Layers.Dropout(0.2))
    decoder.add(Layers.Conv2DTranspose(filters=3, kernel_size=(3, 3), activation='elu', padding='same'))


    return encoder, decoder
```

4. Training the model

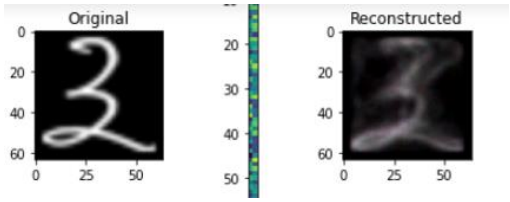
```
from keras.preprocessing.image import ImageDataGenerator

sess = reset_tf_session() #reseting the session, just in case
encoder1,decoder1 = covolutional_autoencoder(IMG_SIZE, code_size=128)

input_l = Layers.Input(IMG_SIZE)
code = encoder1(input_l)
reconstruction = decoder1(code)

optimizer = keras.optimizers.Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

autoencoder1 = keras.models.Model(inputs=input_l, outputs=reconstruction)
autoencoder1.compile(optimizer='rmsprop', loss='mse')

aug = ImageDataGenerator(rotation_range=20, zoom_range=0.15,
    width_shift_range=0.2, height_shift_range=0.2, shear_range=0.15,
    horizontal_flip=True, fill_mode="nearest")

autoencoder1.fit_generator(aug.flow(X_train, X_train),
                validation_data=[X_test, X_test],
                steps_per_epoch=len(X_train),
                epochs=20)
```

5. Visualizing the output

```
autoencoder2 = keras.models.load_model("cnn_autoencoder2.hd5")
encoder2 = keras.models.load_model("cnn_encoder2.hd5")
decoder2 = keras.models.load_model("cnn_decoder2.hd5")
score = autoencoder2.evaluate(X_test,X_test,verbose=0)
print("PCA MSE:", score)

for i in range(5):
    idx = np.random.randint(0,high=(len(X_train)),size=1)[0]
    img = X_train[idx]
    visualize(img,encoder2,decoder2)
```



[1]



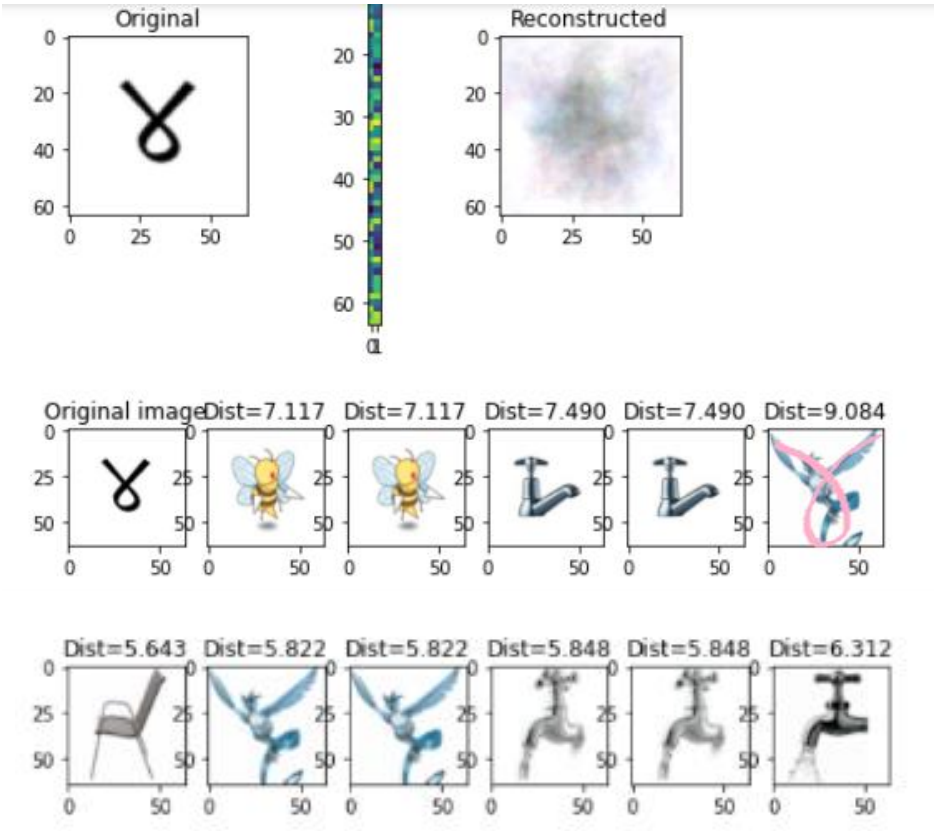
[1] The clarity of the reconstructed output depends on the number of epochs it is trained for. The optimum number identified by this research was 20, however, the image was captured with 5 epochs.

3.4.3 Using Autoencoder II with Hindi font characters

The Autoencoder trained with Mnemonic dataset was fed the Hindi font characters to obtain the Mnemonics.

```
for img_address in Path("D:/images/").glob("**/*.png"):
    addresses.append(img_address)
    img = imageio.imread(img_address, as_gray=False,  pilmode="RGBA")
    img[:,:,0:3][img[:,:,3]==0] = 255
    img = img[:,:,0:3] / 255 - 0.5 #normalizing
    img = transform.resize(img, (64, 64))
    visualize(img,encoder2,decoder2)
    show_similar(img)
```

The output would be similar to the snippet below: -



The images shown are the suggestions for Mnemonic images for the character. The learner can choose whichever is more impactful for them to learn the character size and structure.

4 References

Dataset 1 reference: ccs96307. (2020, 5 11). *Github*. Retrieved from Github: https://github.com/ccs96307/font-to-png

Dataset 2: Jha, S. (2018). *Devnagri Hindi Dataset.* Retrieved from https://www.kaggle.com/jhashanku007/devnagri-hindi-dataset

Dataset 3 (partial): Roy, P., Ghosh, S., & SaumikBhattacharya. (2018). Natural Images. Kaggle.

Dataset 3 (partial): Zhang, L. (2019). 7,000 Labeled Pokemon. Kaggle.

CNN code reference: https://www.edureka.co/blog/convolutional-neural-network/

Autoencoder code reference: https://tech.sc0ville.com/articles/7