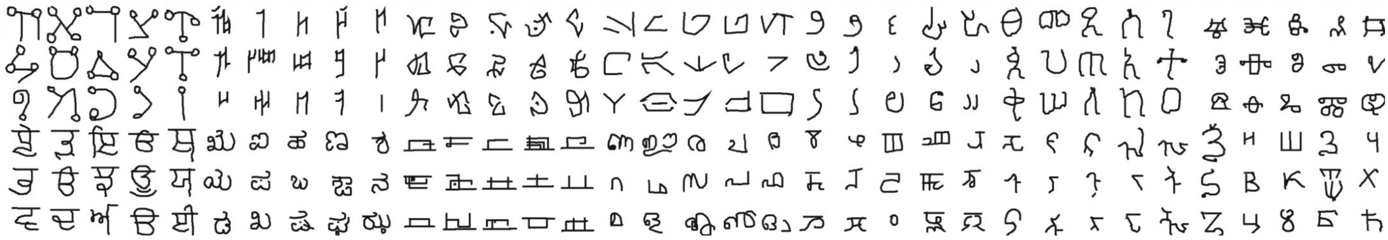
Siamese Neural Networks for One-shot Image Recognition



Importing Dependencies

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random
import pickle as pkl
import cv2
import h5py

from preprocess_data import dataloader

from tqdm import tqdm
from math import ceil

from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from skimage.transform import rotate, AffineTransform, warp, rescale
from skimage.util import random_noise

import tensorflow as tf
from tensorflow.keras import Model, Sequential
from tensorflow.keras.layers import Lambda, Input, Flatten, Dense, Concatenate, Conv2D, Max
from tensorflow.keras.initializers import RandomNormal
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import model_from_json
from scipy.interpolate import make_interp_spline, BSpline
import tensorflow.keras.backend as K
```

In [2]:

```
np.random.seed(0)
random.seed(0)
tf.random.set_seed(0)
```

Plot Training Accuracy and Training Loss

In [3]:

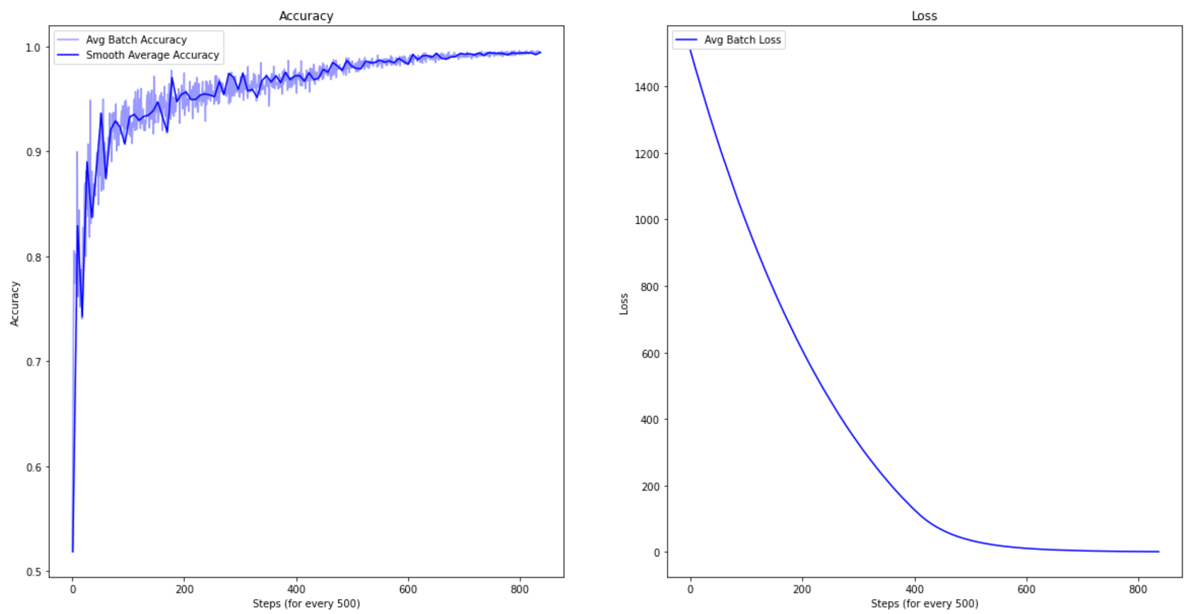
```
def plot_metric(loss, acc):

    size = len(acc)
    x = np.array(range(1,size+1))
    xnew = np.linspace(1,size,100)

    spl = make_interp_spline(x, acc, k=3) #BSpline object
    ynew = spl(xnew)
    plt.figure(figsize=(20,10))
    plt.subplot(1,2,1)
    plt.plot(acc,color = 'b', alpha=0.4, label = 'Avg Batch Accuracy')
    plt.plot(xnew, ynew,color = 'b', alpha=1, label = 'Smooth Average Accuracy')
    plt.xlabel('Steps (for every 500)')
    plt.ylabel('Accuracy')
    plt.legend(loc = "upper left")
    plt.title('Accuracy')

    spl = make_interp_spline(x, loss, k=3) #BSpline object
    ynew = spl(xnew)
    plt.subplot(1,2,2)
    plt.plot(loss,color = 'b', label = 'Avg Batch Loss')
    #plt.plot((xnew, ynew),color = 'b', alpha=1)
    plt.xlabel('Steps (for every 500)')
    plt.ylabel('Loss')
    plt.legend(loc = "upper left")
    plt.title('Loss')
```

In [4]:

```
with open('val_acc','rb') as f:
    v_acc,train_metrics = pkl.load(f)
```

In [5]:

```
train_metrics = np.array(train_metrics)
plot_metric(train_metrics[:,0],train_metrics[:,1])
```



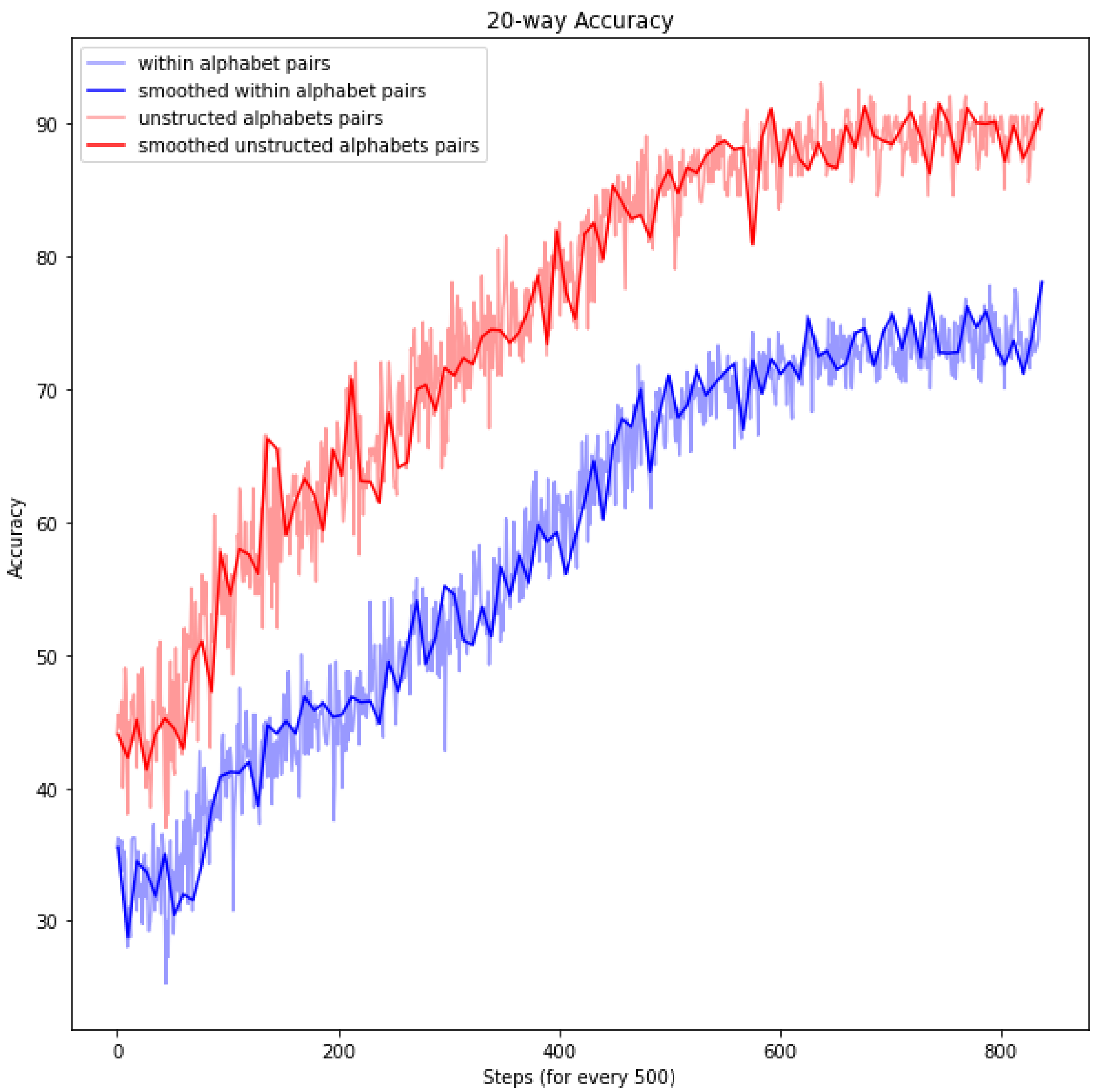
Plotting Validation Accuracy on 20 - way one shot

In [6]:

```
v_acc  =np.array(v_acc)

x = np.array(range(1,len(v_acc)+1))
xnew = np.linspace(1,len(v_acc),100)

spl = make_interp_spline(x, v_acc[:,0], k=3) #BSpline object
ynew1 = spl(xnew)
spl = make_interp_spline(x, v_acc[:,1], k=3) #BSpline object
ynew2 = spl(xnew)
plt.figure(figsize=(10,10))
plt.plot(v_acc[:,0], color = 'b', alpha=0.4,  label = 'within alphabet pairs')
plt.plot(xnew, ynew1, color = 'b', alpha=1, label = 'smoothed within alphabet pairs')
plt.plot(v_acc[:,1], color = 'r', alpha=0.4,  label = 'unstructed alphabets pairs')
plt.plot(xnew, ynew2, color = 'r', alpha=1, label = 'smoothed unstructed alphabets pairs')
plt.xlabel('Steps (for every 500)')
plt.ylabel('Accuracy')
plt.legend(loc = "best")
plt.title('20-way Accuracy')
```

Out[6]:

```
Text(0.5, 1.0, '20-way Accuracy')
```



In [8]:

```
train_metrics[-1]
```

Out[8]:

```
array([1.1560286, 0.9944375])
```

In [9]:

```
v_acc[-1]
```

Out[9]:

```
array([78., 91.])
```

Accuracy on Training Set

Training Accuracy : 99.4 %

Training Loss : 1.15

20-way Accuracy on validation Set

Within Alphabet Pairs Accuracy : 78.0 %

Unstructured Alphabet Pairs Accuracy : 91.0 %

Testing N- way one shot

Loading Saved Model

In [10]:

```
from siamese_train import siamese_network
```

In [11]:

```
siamese = siamese_network()
siamese.get_model()
```

In [12]:

```
best_model = 'best_model/best_model.h5'
siamese.model.load_weights(best_model)
siamese_net = siamese.model
print("Model loaded from disk")
```

```
Model loaded from disk
```

Evaluation on 10 Alphabets (Evaluation Set)

In [16]:

```
wA_file ='wA_eval_10_split_images.pkl'
uA_file ='uA_eval_10_split_images.pkl'

wA_acc, uA_acc = siamese.test_validation_acc(wA_file, uA_file)
```

In [17]:

```
print("Within Alphabet pairs Accuracy for 20-way one shot samples : {}%".format(wA_acc))
print("Unstructured Alphabet pairs Accuracy for 20-way one shot samples : {}%".format(uA_ac
```

```
Within Alphabet pairs Accuracy for 20-way one shot samples : 78.0%
Unstructured Alphabet pairs Accuracy for 20-way one shot samples : 89.0%
```

Evaluation on 20 Alphabets (Image Evaluation Folder)

In [18]:

```
wA_file ='wA_eval_20_split_images.pkl'
uA_file ='uA_eval_20_split_images.pkl'

wA_acc, uA_acc = siamese.test_validation_acc(wA_file, uA_file)

print("Within Alphabet pairs Accuracy for 20-way one shot samples : {}%".format(wA_acc))
print("Unstructured Alphabet pairs Accuracy for 20-way one shot samples : {}%".format(uA_ac
```

```
Within Alphabet pairs Accuracy for 20-way one shot samples : 74.75%
Unstructured Alphabet pairs Accuracy for 20-way one shot samples : 88.0%
```

N-way one shot Testing

In [19]:

```
from preprocess_data import dataloader
import os

folder_path = 'images_evaluation'
dir_list = os.listdir(folder_path)
dl = dataloader()
```

We will now use a base model just to compare our model against this model. This is inspired by Soren Bouma's implementation of Nearest Neighbour pairs.

Our Aim is show that results from siamese net are far better than our base model.

In [20]:

```
def nearest_neighbour_correct(pairs,targets):
    """returns 1 if nearest neighbour gets the correct answer for a one-shot task
        given by (pairs, targets)"""
    X_left, X_right = pairs
    L2_distances = np.zeros_like(targets)
    for i in range(len(targets)):
        L2_distances[i] = np.sqrt(abs(np.sum(X_left[i]**2 - X_right[i]**2)))
    if np.argmin(L2_distances) == np.argmax(targets):
        return 1
    return 0
```

In [21]:

```
def test_data_pairs(n_way = 20, wA = True):
    if wA:
        pairs = dl.wA_test_pairs(folder_path = folder_path, dirs = dir_list, savefilename =
    else:
        pairs = dl.uA_test_pairs(folder_path = folder_path, dirs = dir_list, savefilename =

    X,y =pairs
    correct_pred = 0
    nn_correct = 0
    j = 0
    for i in range(0,len(X),n_way):
        X_left, X_right,_y  = X[i: i+n_way,0],X[i: i+n_way,1], y[i : i+n_way]
        X_left, X_right, _y = np.array(X_left), np.array(X_right), np.array(_y)

        correct_pred += siamese.test_one_shot(X_left, X_right, _y)
        nn_correct += nearest_neighbour_correct((X_left, X_right), _y)

    acc =  correct_pred*100/(len(X)/n_way)
    nn_acc = nn_correct*100/(len(X)/n_way)
    return acc, nn_acc
```

In [59]:

```
def one_shot_accuracy():
    within_accuracies = []
    unstructred_accuracies = []
    #[2, 5, 6, 10, 15, 16, 20]
    for i in tqdm(range(2, 21)):
        within_accuracies.append(test_data_pairs(n_way = i, wA = True))
        unstructred_accuracies.append(test_data_pairs(n_way = i, wA = False))
    return within_accuracies ,unstructred_accuracies
```

In [60]:

```
within_accuracies ,unstructred_accuracies = one_shot_accuracy()
```

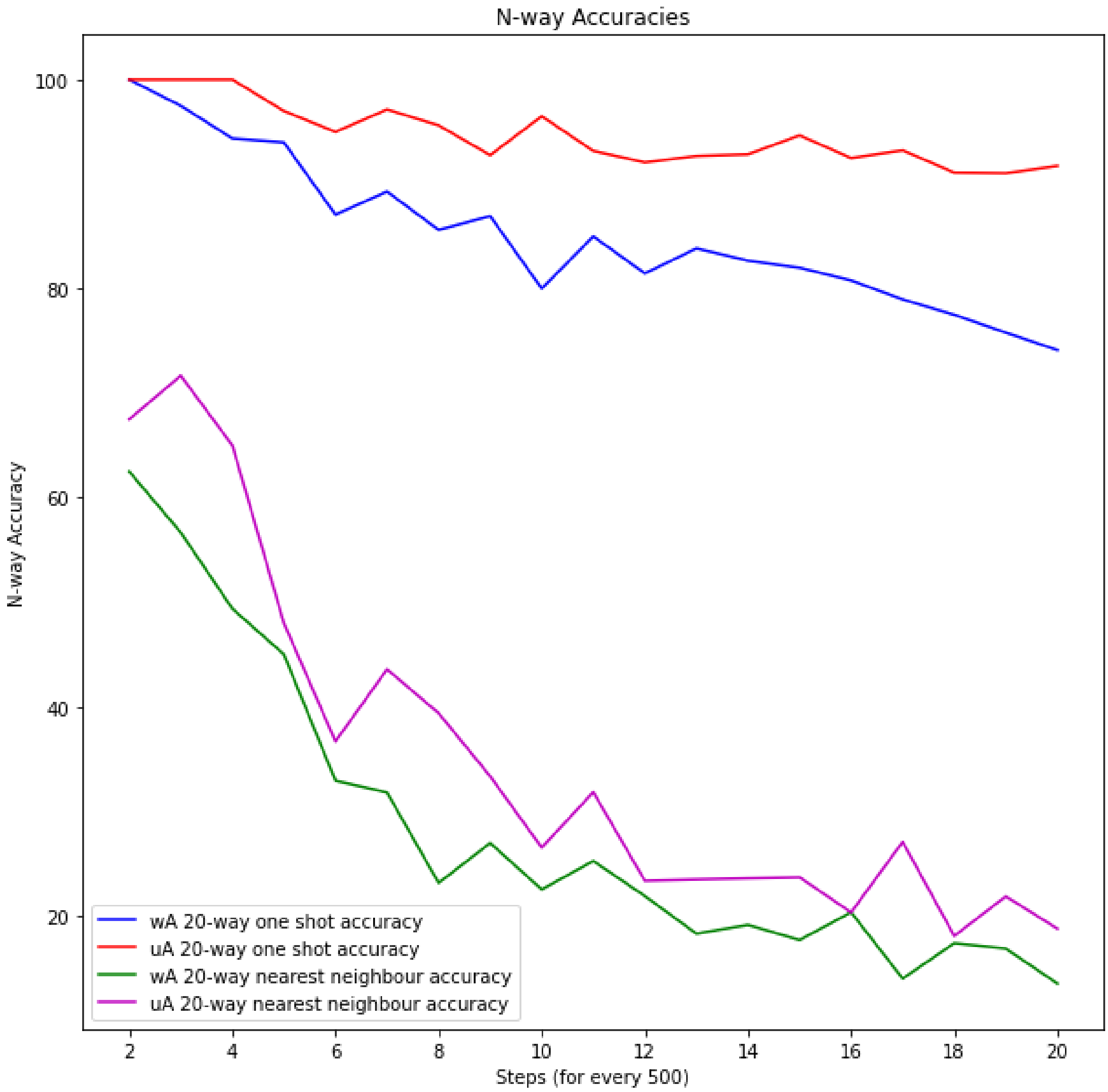
```
100%|████████████████████████████████████████████████████████████
██████████| 19/19 [09:49<00:00, 31.03s/it]
```

In [61]:

```
ways = np.arange(2, 21, 1)

plt.figure(figsize = (10,10))
plt.plot(ways,np.asarray(within_accuracies)[:,0], color = 'b',  label = 'wA 20-way one shot
plt.plot(ways,np.asarray(unstructred_accuracies)[:,0], color = 'r', label = 'uA 20-way one
plt.plot(ways,np.asarray(within_accuracies)[:,1], color = 'g',  label = 'wA 20-way nearest
plt.plot(ways,np.asarray(unstructred_accuracies)[:,1], color = 'm', label = 'uA 20-way near
plt.xlabel('Steps (for every 500)')
plt.xticks(np.arange(2, 22, step=2))
plt.ylabel('N-way Accuracy')
plt.legend(loc = "best")
plt.title('N-way Accuracies')
```

Out[61]:

```
Text(0.5, 1.0, 'N-way Accuracies')
```



In [62]:

```
print(within_accuracies)
```

```
[(100.0, 62.5), (97.5, 56.666666666666664), (94.375, 49.375), (94.0, 45.0),
(87.08333333333333, 32.916666666666664), (89.28571428571429, 31.785714285714
285), (85.625, 23.125), (86.94444444444444, 26.944444444444443), (80.0, 22.
5), (85.0, 25.227272727272727), (81.45833333333333, 21.875), (83.84615384615
384, 18.26923076923077), (82.67857142857143, 19.107142857142858), (82.0, 17.
666666666666668), (80.78125, 20.3125), (78.97058823529412, 13.97058823529411
8), (77.5, 17.36111111111111), (75.78947368421052, 16.842105263157894), (74.
125, 13.5)]
```

In [63]:

```
print(unstructred_accuracies)
```

```
[(100.0, 67.5), (100.0, 71.66666666666667), (100.0, 65.0), (97.0, 48.0), (9
5.0, 36.666666666666664), (97.14285714285714, 43.57142857142857), (95.625, 3
9.375), (92.77777777777777, 33.333333333333336), (96.5, 26.5), (93.181818181
81819, 31.818181818181817), (92.08333333333333, 23.333333333333332), (92.692
3076923077, 23.46153846153846), (92.85714285714286, 23.571428571428573), (9
4.66666666666667, 23.666666666666668), (92.5, 20.3125), (93.23529411764706,
27.058823529411764), (91.11111111111111, 18.055555555555557), (91.0526315789
4737, 21.842105263157894), (91.75, 18.75)]
```

Visualizing N-way pairs

In [25]:

```
from mpl_toolkits.axes_grid1 import ImageGrid
import re
```

In [26]:

```
def generate_img_matrix(X_left, X_right, y):
    X_left, X_right, _y = np.array(X_left), np.array(X_right), np.array(y)
    pred = siamese_net.predict([X_left, X_right])
    index = np.argmax(pred)

    img0 = np.squeeze(X_left[0], axis = 2)
    X_p = []
    img_matrix = []

    for i in range(len(X_right)):
        img1 = np.squeeze(X_right[i], axis = 2)
        X_p.append(img1)
        if len(X_p) == 5:
            X_p =np.vstack(X_p)
            img_matrix.append(X_p)
            X_p = []

    img_matrix = np.asarray(img_matrix)
    img_matrix = np.hstack(img_matrix)
    return img0, img_matrix, index
```

In [27]:

```
def visualize_n_way(file, n_way = 20):

    with open(file,'rb') as f:
        X,y = pkl.load(f)

    i = random.randint(0,int(len(X)/n_way))

    X_left, X_right,_y  = X[i: i+n_way,0],X[i: i+n_way,1], y[i : i+n_way]
    img0, img_matrix, index= generate_img_matrix(X_left, X_right, _y)

    f, ax=  plt.subplots(1,3, figsize = (20,20))
    f.tight_layout()
    ax[0].imshow(img0, cmap = 'gray')
    ax[0].set_title('Test Image')
    ax[1].imshow(img_matrix, cmap = 'gray')
    ax[1].set_title('Support Set')
    ax[2].imshow(np.squeeze(X_right[index], axis = 2), cmap = 'gray')
    ax[2].set_title('Image with highest similarity in Support Set')
```

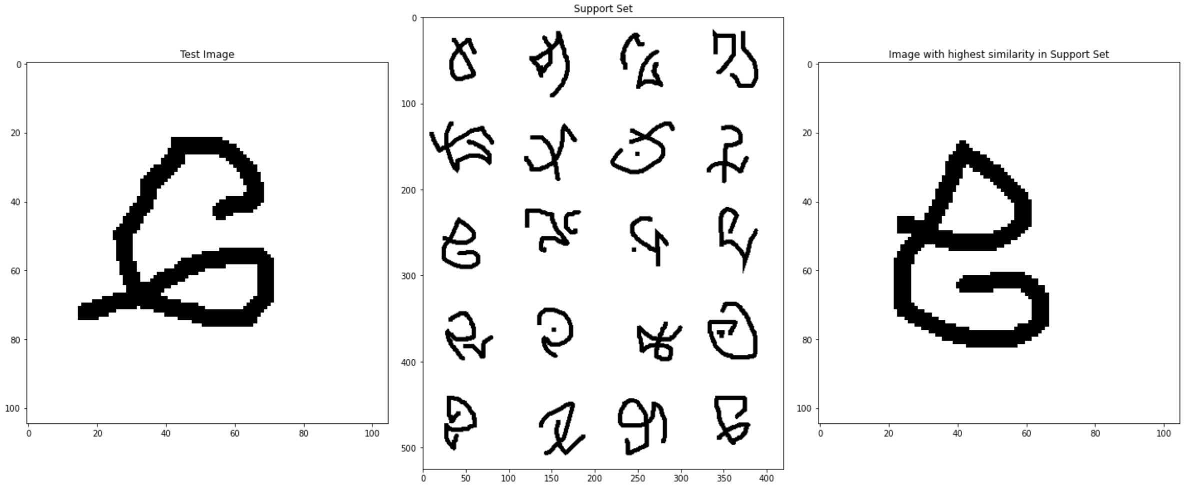
In [28]:

```
wA_file ='wA_eval_20_split_images.pkl'
uA_file ='uA_eval_20_split_images.pkl'
```

Visualizing Within Alphabet Pairs

In [30]:

```
visualize_n_way(wA_file,n_way = 20)
```



Visualizing Unstructured Alphabet Pairs

In [39]:

```
visualize_n_way(uA_file,n_way = 20)
```