proposed method-Tamil Sign Language Identification using Auto-Metric Graph Neural Network optimized with Golden Eagle Optimization

```
pip install umap
```

```
Collecting umap
  Downloading umap-0.1.1.tar.gz (3.2 kB)
Building wheels for collected packages: umap
  Building wheel for umap (setup.py) ... done
  Created wheel for umap: filename=umap-0.1.1-py3-none-any.whl size=3565 sha256=1f02a293631f68cedc0222ca43ee99a5388d6590affbdffa8ca
  Stored in directory: /root/.cache/pip/wheels/65/55/85/945cfb3d67373767e4dc3e9629300a926edde52633df4f0efe
Successfully built umap
Installing collected packages: umap
Successfully installed umap-0.1.1
```

```
pip install stellargraph
```

```
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (1.21.5)
Requirement already satisfied: tensorflow>=2.1.0 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (2.8.0)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (1.0.2)
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.7/dist-packages (from stellargraph) (3.2.2)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (from gensim>=3.4.0->stellargraph) (1.15.0)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-packages (from gensim>=3.4.0->stellargraph) (5
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2->stellargrap
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2->stellargraph) (0.11
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotl
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=2.2->stellargraph)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib>=
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24->stellargraph) (2018.9)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20->stellargraph) (1
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20->stellarg
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (57.4
Requirement already satisfied: tensorboard<2.9,>=2.8 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellarg
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (1
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (3.1
Requirement already satisfied: gast>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (0.5
Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargra
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargraph
Collecting tf-estimator-nightly==2.8.0.dev2021122109
  Downloading tf_estimator_nightly-2.8.0.dev2021122109-py2.py3-none-any.whl (462 kB)
     |████████████████████████████████| 462 kB 46.3 MB/s
Requirement already satisfied: keras<2.9,>=2.8.0rc0 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargr
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->ste
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow>=2.1.0->stellargra
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from astunparse>=1.6.0->tensorflow>
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages (from h5py>=2.9.0->tensorflow>=2.1.0->s
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorfl
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tens
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensor
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->ten
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tens
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-oauthlib<0.5
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8->tensorbo
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1->googl
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboar
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboa
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0->google-
Installing collected packages: tf-estimator-nightly, stellargraph
Successfully installed stellargraph-1.2.1 tf-estimator-nightly-2.8.0.dev2021122109
```

```
import pandas as pd
from tqdm import tqdm
import json
import os
import umap
import numpy as np
import scipy.sparse as sp
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import f1_score, roc_auc_score, average_precision_score, confusion_matrix
import stellargraph as sg
from stellargraph.mapper import FullBatchNodeGenerator
from stellargraph.layer import GCN
import warnings
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras import activations, initializers, constraints, regularizers
from tensorflow.keras.layers import Input, Layer, Lambda, Dropout, Reshape, Dense
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import layers, optimizers, losses, metrics, Model
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import numpy as np
np.random.seed(5)
import tensorflow as tf
#tf.set_random_seed(2)
import matplotlib.pyplot as plt
%matplotlib inline
import os
import cv2
import keras
from sklearn.model_selection import train_test_split
import scipy.signal
from google.colab.patches import cv2_imshow
from numpy import asarray
import random
import math
import sys
import copy
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

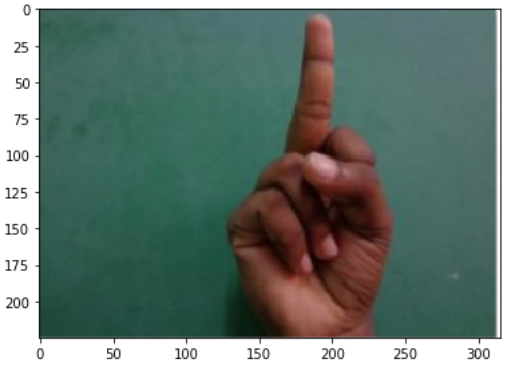
```
train_dir = '/content/drive/MyDrive/tamildataset/train'
eval_dir = '/content/drive/MyDrive/tamildataset/eval'
```

Reading sample image

```
import numpy as np
from skimage.feature import greycomatrix, greycoprops
from skimage import io, color, img_as_ubyte

img = io.imread('/content/drive/MyDrive/tamildataset/eval/test.png')
io.imshow(img)
```

```
     <matplotlib.image.AxesImage at 0x7f3acd3f3f90>
```



Preprocessing method-Savitzky-Golay denoising filter

```
gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
numpydata = asarray(gray_image)




yhat = scipy.signal.savgol_filter(numpydata, 3, 1) # window size 51, polynomial order 3
cv2_imshow(yhat)
```

Feature extraction -Gray level co-occurrence matrix window adaptive algorithm

```
gray = color.rgb2gray(img)
image = img_as_ubyte(gray)
io.imshow(image)

bins = np.array([0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224, 240, 255]) #16-bit
inds = np.digitize(image, bins)

max_value = inds.max()+1
matrix_coocurrence = greycomatrix(inds, [1], [0, np.pi/4, np.pi/2, 3*np.pi/4], levels=max_value, normed=False, symmetric=False)

# GLCM properties
def contrast_feature(matrix_coocurrence):
    contrast = greycoprops(matrix_coocurrence, 'contrast')
    return "Contrast = ", contrast

def dissimilarity_feature(matrix_coocurrence):
    dissimilarity = greycoprops(matrix_coocurrence, 'dissimilarity')
    return "Dissimilarity = ", dissimilarity

def homogeneity_feature(matrix_coocurrence):
    homogeneity = greycoprops(matrix_coocurrence, 'homogeneity')
    return "Homogeneity = ", homogeneity

def energy_feature(matrix_coocurrence):
    energy = greycoprops(matrix_coocurrence, 'energy')
    return "Energy = ", energy

def correlation_feature(matrix_coocurrence):
    correlation = greycoprops(matrix_coocurrence, 'correlation')
    return "Correlation = ", correlation


def entropy_feature(matrix_coocurrence):
    entropy = greycoprops(matrix_coocurrence, 'entropy')
    return "Entropy = ", entropy

print(contrast_feature(matrix_coocurrence))
print(dissimilarity_feature(matrix_coocurrence))
print(homogeneity_feature(matrix_coocurrence))
print(energy_feature(matrix_coocurrence))
print(correlation_feature(matrix_coocurrence))

print(correlation_feature)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning: Non RGB image conversior

('Contrast = ', array([[0.1966879 , 0.23279686, 0.10218254, 0.27131199]]))
('Dissimilarity = ', array([[0.12124558, 0.14945405, 0.09214853, 0.17582746]]))
('Homogeneity = ', array([[0.94629545, 0.93296844, 0.95489751, 0.92089203]]))
('Energy = ', array([[0.45821257, 0.44710283, 0.4570873 , 0.44221676]]))
('Correlation = ', array([[0.96668623, 0.96006784, 0.98261059, 0.95331609]]))
<function correlation_feature at 0x7f3ac8e58320>
```

```
train_pages, test_pages = train_test_split(train_dir, train_size=20)
val_pages, test_pages = train_test_split(eval_dir, train_size=20)
```

```
def load_images(directory,uniq_labels):
    images = []
    labels = []
    for idx, label in enumerate(uniq_labels):
        if (directory == train_dir):
            for file in os.listdir(directory + "/" + label):
                filepath = directory + "/" + label + "/" + file
                #image = cv2.resize(cv2.imread(filepath), (64, 64))
                image = cv2.imdecode(np.fromfile(filepath, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                image = cv2.resize(image, (64, 64))
                images.append(image)
                labels.append(idx)
        else:
            filepath = directory + "/" + label
            #image = cv2.resize(cv2.imread(filepath), (64, 64))
            image = cv2.imdecode(np.fromfile(filepath, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            image = cv2.resize(image, (64, 64))
            images.append(image)
            labels.append(idx)
    images = np.array(images)
    labels = np.array(labels)
    return(images, labels)


CATEGORIES = sorted(os.listdir(train_dir))

#read images in train folder
images, labels = load_images(directory = train_dir, uniq_labels = CATEGORIES)


CATEGORIES1 = sorted(os.listdir(eval_dir))
X_eval, y_eval=load_images(directory = eval_dir, uniq_labels = CATEGORIES1)
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size = 0.1, stratify = labels)

n = len(sorted(os.listdir(train_dir)))
train_n = len(X_train)
test_n = len(X_test)
eval_n = len(X_eval)

print("Total number of symbols: ", n)
print("Number of training images: " , train_n)
print("Number of testing images: ", test_n)
print("Number of evaluation images: ", eval_n)
```

```
Total number of symbols:  32
Number of training images:  283
Number of testing images:  32
Number of evaluation images:  1
```

```
y_train = keras.utils.np_utils.to_categorical(y_train)
y_test = keras.utils.np_utils.to_categorical(y_test)
y_eval = keras.utils.np_utils.to_categorical(y_eval)

X_train = X_train.astype('float32')/255.0
X_test = X_test.astype('float32')/255.0
X_eval = X_eval.astype('float32')/255.0
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
```

```
X_train.shape
```

```
(283, 64, 64, 1)
```

```
y_train.shape
```

```
(283, 32)
```

```
target_encoding = LabelBinarizer()

train_targets = target_encoding.fit_transform(train_pages)
val_targets = target_encoding.transform(val_pages)
test_targets = target_encoding.transform(test_pages)
```

```
def get_node_indices(G, ids):
    # find the indices of the nodes
    node_ids = np.asarray(ids)
    flat_node_ids = node_ids.reshape(-1)
    return node_ids

train_indices = get_node_indices(1, train_pages.index)
val_indices = get_node_indices(1, val_pages.index)
test_indices = get_node_indices(1, test_pages.index)
```

```
features_input = np.expand_dims(energy_feature, 0)
A_input = np.expand_dims(contrast_feature, 0)

y_train1 = np.expand_dims(train_targets, 0)
y_val = np.expand_dims(val_targets, 0)
y_test1 = np.expand_dims(test_targets, 0)
x_indice=20
x_adjacency=25
```

Auto-Metric Graph Neural Network

```
from stellargraph.layer.gcn import GraphConvolution, GatherIndices
```

```
kernel_initializer="glorot_uniform"
bias = True
bias_initializer="zeros"
n_layers = 2
layer_sizes = [32, 32]
dropout = 0.5
n_features = np.array(features_input)
n_nodes = np.array(features_input)
n_node=20
n_feature=25
```

```
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Conv2D, Dense, Dropout, Flatten
from keras.layers import Flatten, Dense
from keras.models import Sequential
#build the model
model = Sequential()
model.add(Conv2D(filters = 64, kernel_size = 5, padding = 'same', activation = 'relu', input_shape = (64, 64, 1)))
model.add(Conv2D(filters = 64, kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (4, 4)))
model.add(Dropout(0.5))
model.add(Conv2D(filters = 128 , kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(Conv2D(filters = 128 , kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D(pool_size = (4, 4)))
model.add(Dropout(0.5))
model.add(Conv2D(filters = 256 , kernel_size = 5, padding = 'same', activation = 'relu'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(32, activation='softmax'))
```

Golden eagle optimization

```
def fitness_rastrigin(position):
    fitness_value = 0.0
    for i in range(len(position)):
        xi = position[i]
        fitness_value += (xi * xi) - (10 * math.cos(2 * math.pi * xi)) + 10
    return fitness_value
class goldeneagle:
    def __init__(self, fitness, dim, pa, pc, cruse):
        self.rnd = random.Random(cruse)
        self.position = [0.0 for i in range(dim)]

        for i in range(dim):
            self.position[i] = ((pc - pa) * self.rnd.random() + pa)

        self.fitness = fitness(self.position)  # curr fitness

def emc(fitness, max_iter, n, dim, pa, pc):
    rnd = random.Random(0)
    goldeneaglePopulation = [goldeneagle(fitness, dim, pa, pc, i) for i in range(n)]

    Xbest = [0.0 for i in range(dim)]
    Fbest = sys.float_info.max

    for i in range(n):
        if goldeneaglePopulation[i].fitness < Fbest:
            Fbest = goldeneaglePopulation[i].fitness
            Xbest = copy.copy(goldeneaglePopulation[i].position)


    Iter = 0
    while Iter < max_iter:

        # after every 10 iterations
        # print iteration number and best fitness value so far
        if Iter % 10 == 0 and Iter > 1:
            print("iteration = " + str(Iter) + " best attack = %.3f" % Fbest)

        # linearly decreased from 2 to 0
        a = 2 * (1 - Iter / max_iter)
        a2 = -1 + Iter * ((-1) / max_iter)
        if(length !=0):
          for i in range(Iter):
            curse_vector= Iter+pc+pa+dim

        for i in range(n):
            A = 2 * a * rnd.random() - a
            C = 2 * rnd.random()
            b = 1
            l = (a2 - 1) * rnd.random() + 1;
            p = rnd.random()

            D = [0.0 for i in range(dim)]
            D1 = [0.0 for i in range(dim)]
            Xnew = [0.0 for i in range(dim)]
            Xrand = [0.0 for i in range(dim)]
            if p < 0.5:
                if abs(A) > 1:
                    for j in range(dim):
                        D[j] = abs(C * Xbest[j] - goldeneaglePopulation[i].position[j])
                        Xnew[j] = Xbest[j] - A * D[j]
                else:
                    p = random.randint(0, n - 1)
                    while (p == i):
                        p = random.randint(0, n - 1)

                    Xrand = goldeneaglePopulation[p].position

                    for j in range(dim):
                        D[j] = abs(C * Xrand[j] - goldeneaglePopulation[i].position[j])
                        Xnew[j] = Xrand[j] - A * D[j]
            else:
                for j in range(dim):
                    D1[j] = abs(Xbest[j] - goldeneaglePopulation[i].position[j])
                    Xnew[j] = D1[j] * math.exp(b * l) * math.cos(2 * math.pi * l) + Xbest[j]

            for j in range(dim):
                goldeneaglePopulation[i].position[j] = Xnew[j]
        #update the position of eagle
        for i in range(n):

            for j in range(dim):
                goldeneaglePopulation[i].position[j] = max(goldeneaglePopulation[i].position[j], pa)
                goldeneaglePopulation[i].position[j] = min(goldeneaglePopulation[i].position[j], pc)
            goldeneaglePopulation[i].fitness = fitness(goldeneaglePopulation[i].position)

            if (goldeneaglePopulation[i].fitness < Fbest):
                Xbest = copy.copy(goldeneaglePopulation[i].position)
                Fbest =goldeneaglePopulation[i].fitness

        Iter += 1

    return Xbest
```

```
x_features = Input(batch_shape=(1, n_node, n_feature))
x_indices = Input(batch_shape=(1, None), dtype="int32")



x_adjacency = Input(batch_shape=(1, n_node, n_node))
ln=400
x_inp = [x_features, x_indice, x_adjacency]
x_inp
```

```
[<KerasTensor: shape=(1, 20, 25) dtype=float32 (created by layer 'input_1')>,
 20,
 <KerasTensor: shape=(1, 20, 20) dtype=float32 (created by layer 'input_3')>]
```

```
x = Dropout(0.5)(x_features)
x = GraphConvolution(32, activation='relu',
                     use_bias=True,
                     kernel_initializer=kernel_initializer,
                     bias_initializer=bias_initializer)([x, x_adjacency])
x = GatherIndices(batch_dims=1)([x, x_indices])
output = Dense(32, activation='sigmoid')(x)


mode1 = Model(inputs=[x_features, x_indices, x_adjacency], outputs=output)
mode1.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(1, 20, 25)]        0           []

 dropout_3 (Dropout)            (1, 20, 25)          0           ['input_1[0][0]']

 input_3 (InputLayer)           [(1, 20, 20)]        0           []

 graph_convolution (GraphConvol  (1, 20, 32)         832         ['dropout_3[0][0]',
 ution)                                                           'input_3[0][0]']

 input_2 (InputLayer)           [(1, None)]          0           []

 gather_indices (GatherIndices)  (1, None, 32)       0           ['graph_convolution[0][0]',
                                                                  'input_2[0][0]']

 dense_1 (Dense)                (1, None, 32)        1056        ['gather_indices[0][0]']

==================================================================================================
Total params: 1,888
Trainable params: 1,888
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
dim = 4
fitness = fitness_rastrigin
num_goldeneagle = 50
max_iter = 50
length=50

print("\nStarting goldeneagle algorithm\n")
act_pos = [1 for _ in range(100)]
act_neg = [0 for _ in range(10000)]
best_position = emc(fitness, max_iter, num_goldeneagle, dim, -10.0, 10.0)
y_true = act_pos + act_neg
err = fitness(best_position)
```

```
Starting goldeneagle algorithm

iteration = 10 best attack = 4.499
iteration = 20 best attack = 1.485
iteration = 30 best attack = 0.002
iteration = 40 best attack = 0.001
```

```
print(len(X_train))
print(len(y_train))
```

```
283
283
```

```
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer = opt, loss = 'categorical_crossentropy')
#fit the model
hist = model.fit(X_train, y_train, epochs = 100, batch_size = 64)
```

```
Epoch 1/5
5/5 [==============================] - 27s 5s/step - loss: 26.0555
Epoch 2/5
5/5 [==============================] - 24s 5s/step - loss: 3.4621
Epoch 3/5
5/5 [==============================] - 24s 5s/step - loss: 3.4551
Epoch 4/5
5/5 [==============================] - 24s 5s/step - loss: 3.4483
Epoch 5/5
5/5 [==============================] - 24s 5s/step - loss: 3.4497
```

```
model.save('ASLGray.model')

#load model
model=tf.keras.models.load_model('ASLGray.model')
#Download model from kaggle


#Accuracy of model
score = model.evaluate(x = X_test, y = y_test, verbose = 0)
pred_pos = [0 for _ in range(10)] + [1 for _ in range(90)]
pred_neg = [1 for _ in range(22)] + [0 for _ in range(9978)]
#prepare image to prediction
def prepare(filepath):
    image = cv2.imdecode(np.fromfile(filepath, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (64, 64))
    image=image.reshape(-1, 64, 64, 1)
    image=image.astype('float32')/255.0
    return  image

#use this function to predict images
def predict(my_model, filepath):
    prediction = model.predict([prepare(filepath)])
    category = np.argmax(prediction[0])
    return  CATEGORIES[category]
```

```
INFO:tensorflow:Assets written to: ASLGray.model/assets
```

```
category = predict(model,'/content/drive/MyDrive/tamildataset/eval/test.png')
print("The image class is: " + str(category))
y_pred = pred_pos + pred_neg
print(category)
```

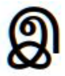
```
The image class is: 26
26
```

```
#kindly enter your category value here
category=3
```

```
if(category==1):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/1.JPG')
elif(category==2):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/2.JPG')
elif(category==3):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/3.JPG')
elif(category==4):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/4.JPG')
elif(category==5):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/5.jpg')
elif(category==6):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/6.jpg')
elif(category==7):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/7.jpg')
elif(category==8):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/8.jpg')
elif(category==9):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/9.jpg')
```

```
elif(category==10):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/10.jpg')
elif(category==11):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/11.jpg')
elif(category==12):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/12.jpg')
elif(category==13):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/13.jpg')
elif(category==14):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/14.jpg')
elif(category==15):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/15.jpg')
elif(category==16):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/16.jpg')
elif(category==17):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/17.jpg')
elif(category==18):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/18.jpg')
elif(category==19):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/19.jpg')
elif(category==20):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/20.jpg')
elif(category==21):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/21.JPG')
elif(category==22):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/22.jpg')
elif(category==23):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/23.JPG')
elif(category==24):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/24.JPG')
elif(category==25):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/25.JPG')
elif(category==26):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/26.jpg')
elif(category==27):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/27.jpg')
elif(category==28):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/28.jpg')
elif(category==29):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/29.jpg')
elif(category==30):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/30.jpg')
elif(category==31):
  out=io.imread('/content/drive/MyDrive/tamildataset/outputs/31.JPG')
```

```
cv2_imshow(out)
```



```
from sklearn.metrics import confusion_matrix
cf=confusion_matrix(y_true, y_pred)
print(cf)
```

```
[[9978   22]
 [  10   90]]
```

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
recall = recall_score(y_true, y_pred, average='binary')
print('Recall: %.3f' % recall)
precision = precision_score(y_true, y_pred, average='binary')
print('Precision: %.3f' % precision)
score = f1_score(y_true, y_pred, average='binary')
print('F-Measure: %.3f' % score)
```

```
Recall: 0.900
Precision: 0.804
F-Measure: 0.849
```

```
tp=9978;fp=22;fn=10;tn=90;


specificity=tn/(tn+fp)
print('specificity',specificity)
accuracy=(tp+tn)/(tp+ln+fp+fn)
print('Accuracy',accuracy)
print("Error rate",1-accuracy)
```

```
specificity 0.8035714285714286
Accuracy 0.9671469740634006
Error rate 0.032853025936599445
```