**M-AI-AN**

**Information**

M-AI-AN is an AI system which can be used to identify Mayan glyphs.

It is trained on glyphs from The Mayan Epigraphic Database1.

**Rationale**

M-AI-AN is a Combination of CNN and RNN neural networks, chained together, to provide an English translation of Mayan Glyphs.

**Methodology:**

M-AI-AN is to be trained to do the following:

1. Identify the sequence of glyphs in a writing block [CNN]

2. Identify each glyph with affixes within the block [CNN]

3. Translate the resulting text into English [RNN]

The main training regimen is as follows:

1. Main glyphs
2. Affixes
3. Face glyphs

**References**

1 <http://www2.iath.virginia.edu/med/index.html>

**Current Progress:**

Main Glyph Identifier:

1. Data Formatter [COMPLETED]

* Using Mayan Epigraphic Database to create test data for the convolutional network
  + Lack of large scale “real world” data
  + Work around by using graphical representation of glyphs with edits
    - Performs mild translations and rotations to provide input data relevant to input photographs
* Creates a specified number of training samples based on specific glyph indicies.

2. M-AI-AN Classifier [COMPLETED]

* Train model on training and test data
  + Split data into subgroups for training and testing
  + Train for finite number of epochs
* Test multiple convolutional network designs
  + Tested 16 separate designs
  + Kept accuracy logs for each design with log of changes made and relevant calculations for the design of the next
* Implemented testing data image identification
* Implemented custom image data identification
  + Uses Pillow library to modify image to similar black and white “graphic” style as training data
  + Feeds into network to produce output glyph index

Future Progress:

* Usage of multi-head convolutional model to decypher entire Mayan glyphs:
  + Glyphs can have up to six different parts
  + At least six different heads to be created
    - Potential to double up heads for most common patterns (ie. 3 seperate affix/main combinations)
* Place on to GitHub for public viewing

**CODE BELOW**

**FILENAME: M-AI-ANclassifier15.py**

#M-AI-AN Main Program

#Used to train M-AI-AN

#Used to classify images presented to M-AI-AN

#Imports

import torch

import torch.nn as nn

import torch.nn.functional as F

from torch.utils.data import DataLoader

from torchvision import datasets, transforms, models # add models to the list

from torchvision.utils import make\_grid

from PIL import Image, ImageOps

import os

import datetime

import GlyLib as GL

import numpy as np

import pandas as pd

root = os.getcwd()

print("\n")

print("---------------------------------------")

print("@@@ M-AI-AN Classifier 15 @@@")

print("---------------------------------------")

set = 0

while set == 0:

print("Please choose from the following options:")

print("1 - Model Training")

print("2 - Image Classification")

print("3 - Model Details")

print("X - Exit")

choice = input()

#Model Definition

class ConvolutionalNetwork(nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.conv1 = nn.Conv2d(3, 6, 3, 1)

self.conv2 = nn.Conv2d(6, 16, 2, 1)

self.fc1 = nn.Linear(47\*47\*16, 360)

self.fc2 = nn.Linear(360, 120)

self.fc3 = nn.Linear(120, 1138)

def forward(self, X):

#Define dropout

gendropout = nn.Dropout(p=0.1)

detdropout = nn.Dropout(p=0.2)

X = F.relu(self.conv1(X))

X = F.avg\_pool2d(X, 2, 2)

X = gendropout(X)

X = F.relu(self.conv2(X))

X = F.avg\_pool2d(X, 2, 2)

X = detdropout(X)

X = X.view(-1, 47\*47\*16)

X = F.relu(self.fc1(X))

X = F.relu(self.fc2(X))

X = self.fc3(X)

return F.log\_softmax(X, dim=1)

MAiAn = ConvolutionalNetwork()

MAiAn

if choice == "1":

print("-@-----------------------------------@-")

print("Please name model export:")

modelname = input()

#Loss and optimization

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(MAiAn.parameters(), lr=0.001)

scheduler = torch.optim.lr\_scheduler.StepLR(optimizer, step\_size=1, gamma=0.5)

#Training Code

#----------------------------------------------

#Data Import

#Tensorize code

tensorize = transforms.Compose([

transforms.ToTensor()

])

#import data

train\_data = datasets.ImageFolder(root + '\\TestData\\train', transform = tensorize)

test\_data = datasets.ImageFolder(root + '\\TestData\\test', transform = tensorize)

#Define classes

class\_names = test\_data.classes

#Define loaders

train\_loader = DataLoader(train\_data, batch\_size=2, shuffle=True)

test\_loader = DataLoader(test\_data, batch\_size=2, shuffle=False)

#Time stamp code

import time

start\_time = time.time()

#Training settings and variables

epochs = 5

train\_losses = []

test\_losses = []

train\_correct = []

test\_correct = []

#Logging code

traininglog = open('TrainingLog.csv','w')

epoch\_list = []

batch\_list = []

loss\_list = []

accuracy\_list = []

for i in range(epochs):

trn\_corr = 0

tst\_corr = 0

##DEBUGGING CODE

# Run the training batches

for b, (X\_train, y\_train) in enumerate(train\_loader):

b+=1

# Apply the model

y\_pred = MAiAn(X\_train) # we don't flatten X-train here

loss = criterion(y\_pred, y\_train)

# Tally the number of correct predictions

predicted = torch.max(y\_pred.data, 1)[1]

batch\_corr = (predicted == y\_train).sum()

trn\_corr += batch\_corr

# Update parameters

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# Print interim results

if b%200 == 0:

curr\_lr = optimizer.param\_groups[0]['lr']

print(f'Time: {datetime.datetime.now().strftime("%H:%M:%S")} epoch: {i:2} batch: {b:4} [{2\*b:6}/{str(len(train\_data))}] loss: {loss.item():10.8f} accuracy: {trn\_corr.item()\*100/(10\*b):7.3f}% LRate:{curr\_lr}')

epoch\_list.append(i)

batch\_list.append(b)

loss\_list.append(loss.item())

accuracy\_list.append(trn\_corr.item()\*100/(10\*b))

train\_losses.append(loss)

train\_correct.append(trn\_corr)

# Run the testing batches

with torch.no\_grad():

for b, (X\_test, y\_test) in enumerate(test\_loader):

# Apply the model

y\_val = MAiAn(X\_test)

# Tally the number of correct predictions

predicted = torch.max(y\_val.data, 1)[1]

tst\_corr += (predicted == y\_test).sum()

loss = criterion(y\_val, y\_test)

#Scheduler Code

scheduler.step()

test\_losses.append(loss)

test\_correct.append(tst\_corr)

print(f'\nDuration: {time.time() - start\_time:.0f} seconds') # print the time elapsed

#Log training results and export to file

traininglog.write("Training log - " + str(start\_time) + "\n")

traininglog.write("Epoch" + "," + "Batch" + "," + "Loss" + "," + "Accuracy" + "\n")

for pos, val in enumerate(train\_losses):

traininglog.write(str(epoch\_list[pos]) + "," + str(batch\_list[pos]) + "," + str(loss\_list[pos]) + "," + str(accuracy\_list[pos]) + "\n")

traininglog.close()

### EVALUATE Test Data ###

# Extract the data all at once, not in batches

test\_load\_all = DataLoader(test\_data, batch\_size=100, shuffle=False)

with torch.no\_grad():

correct = 0

for X\_test, y\_test in test\_load\_all:

y\_val = MAiAn(X\_test) # we don't flatten the data this time

predicted = torch.max(y\_val,1)[1]

correct += (predicted == y\_test).sum()

print(f'Test accuracy: {correct.item()}/{len(test\_data)} = {correct.item()\*100/(len(test\_data)):7.3f}%')

torch.save(MAiAn.state\_dict(), str(modelname) + ".pt")

print("Training Complete")

print("Model saved as: " +str(modelname) + ".pt" )

print("\n")

print("-@-----------------------------------@-")

if choice == "2":

#Image classifier

print("Select Classification Option:")

print("1 - Individual Test Data Classification")

print("2 - Custom Image")

print("3 - Batch Testing")

print("X - Back to menu")

classifychoice = input()

#Tensorize code

tensorize = transforms.Compose([

transforms.ToTensor()

])

#Load Model

MAiAn.load\_state\_dict(torch.load('M-AI-ANfull.pt'))

if classifychoice == "1":

#Individual test classification

#import test data

test\_data = datasets.ImageFolder(root + '\\TestData\\test', transform = tensorize)

#Define classes

class\_names = test\_data.classes

looper = 0

while looper == 0:

print("Please type the glyph type to use: ")

glyphtypeinput = input()

print("Please type the glyph image to use: ")

individualglyphinput = input()

x = -1

f = 0

n = 0

for folder, subfolders, filenames in os.walk("TestData/test"):

x = x + 1

n = 0

#print("Folder :" + str(folder))

# print(subfolders)

# print("Filenames :" + str(filenames))

if folder.find(glyphtypeinput) > -1:

#print(folder[folder.find(glyphtypeinput):len(folder)])

f = x

#print(str(f))

for image in filenames:

n = n + 1

#Define Data Loader

test\_loader = DataLoader(test\_data, batch\_size=2, shuffle=False)

#Begin Classification

x = ((f - 1) \* n) + int(individualglyphinput)

MAiAn.eval

with torch.no\_grad():

new\_pred = MAiAn(test\_data[x][0].view(1,3,195,195)).argmax()

print(f'Predicted value: {new\_pred.item()} {class\_names[new\_pred.item()]} {GL.IdToWord.get("0" + str(class\_names[new\_pred.item()]))}')

print("\n")

print("\n")

print("-@-----------------------------------@-")

if classifychoice == "2":

customchoice = input("Please give filename without extensions: ")

#Sanitizes custome image for use in classifier

workimage = Image.open(root + '\\CustomImage\\ClassifyArea\\' + customchoice + '.jpg')

editedimage = ImageOps.fit(workimage, [195,195],centering=(0.5, 0.5))

editedimage = ImageOps.autocontrast(editedimage, cutoff=(0.05, 0.95), ignore = None, mask = None)

print(editedimage.size)

#editedimage = ImageOps.grayscale(editedimage)

CustomImage = tensorize(editedimage)

#import test data

test\_data = datasets.ImageFolder(root + '\\TestData\\test', transform = tensorize)

#Define classes

class\_names = test\_data.classes

MAiAn.eval

with torch.no\_grad():

new\_pred = MAiAn(CustomImage.view(1,3,195,195)).argmax()

print(f'Predicted value: {new\_pred.item()} {class\_names[new\_pred.item()]} {GL.IdToWord.get("0" + str(class\_names[new\_pred.item()]))}')

print("\n")

#Print List of probabilities

with torch.no\_grad():

new\_pred = torch.argsort(MAiAn(CustomImage.view(1,3,195,195)), dim=1,descending=True)

print(f'Predicted values list:')

#for i in range(len(new\_pred[0])):

for i in range(0,15):

#print(len(new\_pred[0]))

#print(range(len(new\_pred)))

print("No: " + str(new\_pred[0][i].item()) + " ID: " + str(class\_names[new\_pred[0][i].item()]))

#print(GL.IdToWord.get("0" + str(class\_names[new\_pred[0][i].item()])))

#print(str(new\_pred))

print("\n")

print("-@-----------------------------------@-")

if classifychoice == "3":

#Batch Testing

#import data

test\_data = datasets.ImageFolder(root + '\\TestData\\test', transform = tensorize)

#Tests against a large amount of data

test\_load\_all = DataLoader(test\_data, batch\_size=100, shuffle=False)

#Define classes

class\_names = test\_data.classes

MAiAn.eval

with torch.no\_grad():

correct = 0

for X\_test, y\_test in test\_load\_all:

batch = 0

end = 0

for pos, val in enumerate(X\_test):

end = end + 1

prediction = MAiAn(val.view(1,3,195,195)).argmax()

if prediction.item() == y\_test[pos]:

batch = batch + 1

print("Batch Accuracy: " + str(batch) + "/" + str(end))

correct = correct + batch

print(f'Test accuracy: {correct}/{len(test\_data)}')

print("\n")

print("-@-----------------------------------@-")

if classifychoice == "X":

print("Returning to menu")

print("\n")

print("-@-----------------------------------@-")

if choice == "3":

print(MAiAn)

print("\n")

print("-@-----------------------------------@-")

if choice == "X":

set = 1

**FILENAME: M-AI-ANclassifier15.py**