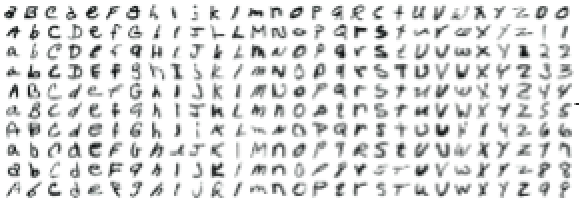
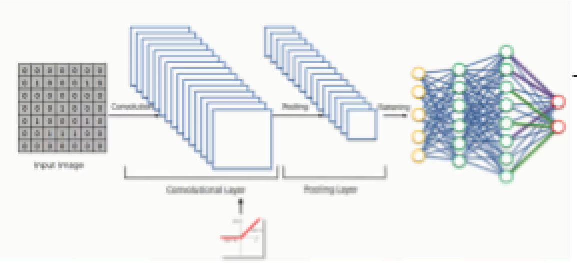
Ryan Taylor   January 23, 2020

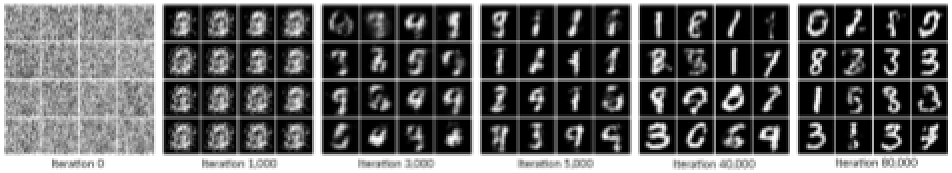
Handwriting Synthesis: Using a CNN, Generative Adversarial Network and YOLOv3 detection-classification algorithm to create handwritten-style versions of digital text



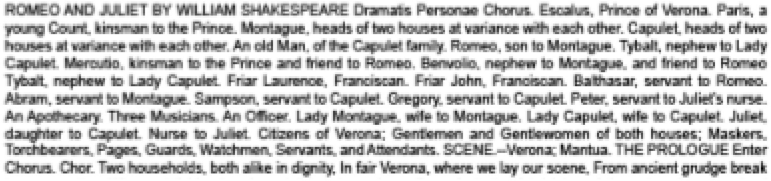
EMNIST Dataset



Convolutional Neural Network



Generative Adversarial Network



ROMEO AND JULIET BY WILLIAM SHAKESPEARE Dramatis Personae Chorus. Escalus, Prince of Verona. Paris, a young Count, kinsman to the Prince. Montague, heads of two houses at variance with each other. Capulet, heads of two houses at variance with each other. An old Man, of the Capulet family. Romeo, son to Montague. Tybalt, nephew to Lady Capulet. Mercutio, kinsman to the Prince and friend to Romeo. Benvolio, nephew to Montague, and friend to Romeo Tybalt, nephew to Lady Capulet. Friar Laurence, Franciscan. Friar John, Franciscan. Balthasar, servant to Romeo. Abram, servant to Montague. Sampson, servant to Capulet. Gregory, servant to Capulet. Peter, servant to Juliet's nurse. An Apothecary. Three Musicians. An Officer. Lady Montague, wife to Montague. Lady Capulet, wife to Capulet. Juliet, daughter to Capulet. Nurse to Juliet. Citizens of Verona; Gentlemen and Gentlewomen of both houses; Maskers, Torchbearers, Pages, Guards, Watchmen, Servants, and Attendants. SCENE.–Verona; Mantua. THE PROLOGUE Enter Chorus. Chor. Two households, both alike in dignity, In fair Verona, where we lay our scene, From ancient grudge break

Text Input

+

YOLOv3 Real-Time Object Detection

=

Ryan Taylor January 23, 2020

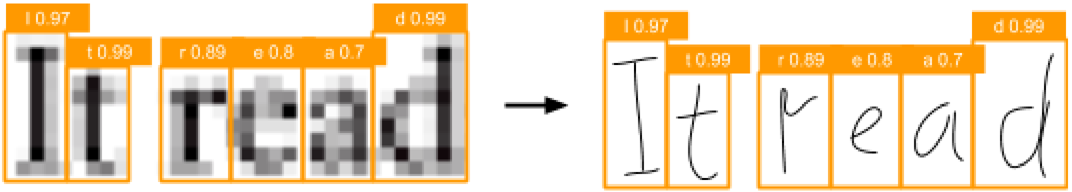
Summary

This project will involve multiple systems of artificial intelligence, including the following:

- Dense Fully-Connected Neural Nets
- Convolution
- Synthesis
    - Generative Adversarial Network
    - Discrimination
- Object Detection
    - YOLOv3 algorithm

Ultimately, the goal of this project will be to produce images of handwritten text given an input image of typed text and 28x28 pixel segments of random grayscale noise.

I will refer to digitally-produced text (below, left) as ‘font’ and handwritten text (below, right) as ‘drawing’ (the predictions are entirely made up):



Progress:

The first step of the project was to classify font-based characters. Doing so required data manipulation on the UCI Character Font Images Dataset, which contains 745,000 instances of 411 attributes, for a total of 306 million data points. The data itself is segmented into .csv files corresponding to individual font styles. Below is an example of ‘ARIAL’ data:

font	fontVariant	m_label	strength	italic	orientation	m_top	m_left	original H	original W	h	w	r0c0	r0c1
ARIAL	scanned	48	0.4	0	0	0	0	15	25	20	20	1	1

ARIAL	scanned	50	0.4	0	0	0	0	18	11	20	20	4	7
ARIAL	scanned	83	0.4	0	0	0	0	13	10	20	20	1	1

The unusual nature of these data required careful handling and design of the import statements. Individual rows are single characters. “m_label” is the ASCII decimal value of the character. The highlighted values are used to identify array indices (r0c0 → (0,0)) The functional code is as follows:

```
file = pd.io.parsers.read_csv(csv)

    imgs = list()
    labels = list()

    total = len(file['font'])

    for i in range(total) :
        if(file['m_label'][i] < 127 and file['strength'][i] == 0.4 and file['italic'][i] == 0) : #check that 
character is a-z, A-Z, 0-9, special chars, reg.
            image = np.empty((20,20))
            sys.stdout.write("\rCreating image %i, %s complete" % (i, str(round((i/total) * 100))))
            sys.stdout.flush()
            for r in range(20) :
                for c in range(20) :
                    val = file["r%sc%d"%(r,c)][i]
                    image.itemset((r,c), val)
            labels.append(file['m_label'][i])
            imgs.append(image)
    return imgs, labels
```

The bolded line uses Python string formatting along with an array index to grab the correct column corresponding to an array value (ex. “r6c10”)

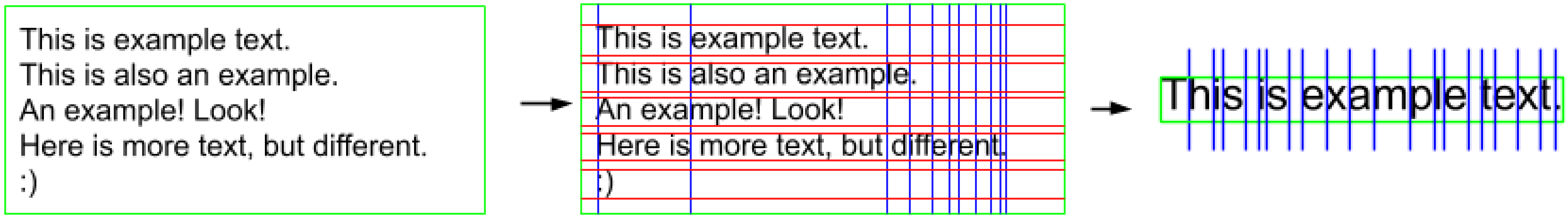
Ryan Taylor January 23, 2020

So far, I have written and trained a convolutional neural network on the CFID dataset with the following layers:

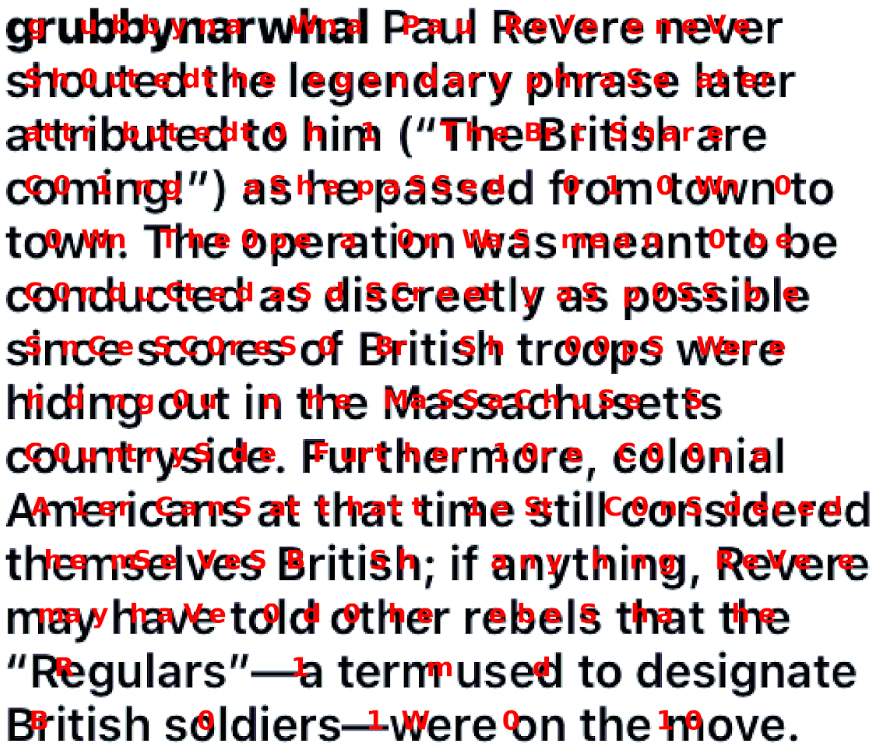
model = Sequential()
Conv2D(32, (2,2), padding='same', name='Conv1', input_shape=(20,20,1), activation='tanh')
BatchNormalization()
Conv2D(32, (2,2), padding='same', name='Conv2', activation='tanh')
BatchNormalization()
Flatten()
Dense(units=512, activation='softmax', name='Dense1', input_shape=(400,))
Dropout(0.25)
Dense(units=classes, activation='softmax', name='Dense2', input_shape=(400,))

The network has a JSON file and an hd5 file that store the model configuration and weights respectively. After a few hours, the network achieved 99.964% accuracy on the training set. This number, while high, is somewhat inflated by overfitting the data; as such, it will not perform as well on less normal characters. But the network largely succeeds in identifying characters on the validation set. Training has also only been performed on a limited set of data (those characters in the ‘ARIAL.csv’ file) which restricts the number of fonts the CNN can accurately predict.

The next step is to implement a form of near-real-time object detection and classification for images of font-based text. Ideally, such implementation would occur through the use of YOLOv3 or a specially-designed RCNN; however, such implementations can be exceedingly difficult, especially with an existing architecture that is somewhat prone to misclassification. Additionally, the nature of input data is such that it is already divided into distinct rows and columns, much like this paragraph. With this in mind, I designed a ‘low-level’ array reading and writing architecture that uses a set of functions that divide and trim input data recursively before performing classification. An example is below:



The recursion breaks once the furthest subdivision has been identified (no vertical or horizontal divisions can be made). At that point, the character is stored in a Box object, which saves the initial index of the bounding box around the character relative to the entire image, the length and width of the character, an array of the contents of the character lifted, and a method that gets a prediction for the character given a model parameter. This process was implemented and tested on an image of an Instagram comment about Paul Revere (No, I don’t remember why I have this image):



grubbynarwhal Paul Revere never
shouted the legendary phrase later
attributed to him (“The British are
coming!”) as he passed from town to
town. The operation was meant to be
conducted as discreetly as possible
since scores of British troops were
hiding out in the Massachusetts
countryside. Furthermore, colonial
Americans at that time still considered
themselves British; if anything, Revere
may have told other rebels that the
“Regulars”—a term used to designate
British soldiers—were on the move.

Ryan Taylor   January 23, 2020

The red characters are predictions, positioned closely to each respective character. The characters that are missing are due to a failure of the detection algorithm to find a character, rather than a failure of the neural network to identify one. Unfortunately, this algorithm was optimized to work only with this one example, and does not succeed to the same extent as above when given other data. I am currently improving this code by starting from scratch and reimplementing portions of code after they have been tested on multiple input sources. The full code is below.

Object Detection & Classification

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 16 17:33:18 2020

@author: ryan
"""

# classifier w/ superimposition

import numpy as np

################  Keras NN Code  ################

def get_scaled_image(imgpath) :
    import cv2
    from skimage.io import imread

    img = imread(imgpath, as_gray = True)
    #img = np.asarray(img, dtype='float32')
    newimg = cv2.resize(img, (20,20))
    newimg = np.reshape(newimg, (1,20,20,1))
    return newimg

def get_scaled_arr(im) :
    import cv2
    newimg = im
    #img = np.asarray(img, dtype='float32')
    if( newimg.shape != (20,20)):
        newimg = cv2.resize(img, (20,20))
    newimg = np.reshape(newimg, (1,20,20,1))
    newimg *= 255
    return newimg.astype('uint8')

def get_image_prediction(imgpath, model) :
    test_img = get_scaled_image(imgpath)
    predicts = predict_and_plot(test_img,model)
    return predicts

def predict_and_plot(xtrain, model) :
    predicts = predict_class(xtrain, model)
    #print(predicts)
    predicts = np.asarray(predicts)
    plot_multiple_images(xtrain, predicts)
    return predicts

def plot_multiple_images(lis, labs) :
    image = np.empty((20, 20*len(lis)))
    string = ""
    for c in range(len(lis)) :
        newimg = lis[c]
        string += str(labs[c])
        for x in range(20) :
            for y in range(20) :
                image.itemset((x,(c*20) + y), newimg.item((x,y,0)))
    plot.xlabel(string)
    plot.imshow(image[:,:])

def predict_class(im, model) :
    imga = get_scaled_image(im)
    plot.imshow(np.reshape(imga, (20,20)))
    p = model.predict(imga, batch_size=1, verbose=1)
    p = np.asarray(p, dtype='float32')
    out = list()
    for i in range(len(p)) :
        out.append(chr(33+np.argmax(p[i])))
        #out.append(chr(33+np.nonzero(p[i])[0]))
    return out

def predict_class_arr(imw, model) :
    imagen = imw#(invert(imw))
    plot.imshow(np.reshape(imagen,(20,20)),cmap='Blues')
    imagen = np.reshape(imagen, (1,20,20,1))
    p = model.predict(imagen, batch_size=1, verbose=1)
    p = np.asarray(p, dtype='float32')
    out = list()
    for i in range(len(p)) :
        out.append(chr(33+np.argmax(p[i])))
        #out.append(chr(33+np.nonzero(p[i])[0]))
    return out

def load_model_from_JSON(modelpath, weightpath) :
    from keras.models import model_from_json
    json_file = open(modelpath, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights(weightpath)
    return loaded_model

###########  Geom Obj Classification  ###########

# -*- coding: utf-8 -*-
"""
Created on Mon Jan 13 22:11:31 2020

@author: Ryan
"""
from matplotlib import pyplot as plot

def create_scaled_image(loc, dim) :
    import cv2
    from skimage.io import imread
    img = imread(loc, as_gray = True)
    newimg = cv2.resize(img, dim)
    newimg *= 255
    return newimg.astype('uint8')

def rescale_img(img, dim) :
    import cv2
    newimg = cv2.resize(img, dim)
    return newimg

def get_background_value(img) :
    (values,counts) = np.unique(img,return_counts=True)
    ind=np.argmax(counts)
    return values[ind]

def auto_tolerance(img) :
    avg = np.mean(img,axis=(0,1))
    return (avg/2)

def invert(img) :
    return np.absolute(np.subtract(img, 255))

def get_horizontal_splits(arr, bkg, tolerance) :
    rows = list()
    for row in range(arr.shape[0]) :
        empty = True
        for i in range(arr.shape[1]) :
            if (abs(int(arr[row,i])-bkg) > tolerance) :
                empty = False
        if empty == True :
            rows.append(row)
    rows.append(0)
    rows.append(arr.shape[0])
    return rows
def get_vertical_splits(arr, bkg, tolerance) :
    cols = list()
    for col in range(arr.shape[1]) :
        empty = True
        for i in range(arr.shape[0]) :
            if abs(int(arr[i,col])-bkg) > tolerance :
                empty = False
        if empty == True :
            cols.append(col)
    cols.append(0)
    cols.append(arr.shape[1])
    return cols

def get_bounding_boxes(horizontals,verticals,image) :
    #start at zero for both rows and cols, move until a val in either is encountered, then proceed along other dim
    #for h in horizontals :
    img_hmax = image.shape[0]
    img_vmax = image.shape[1]

    if horizontals.count(0) == 0 :
        horizontals.insert(0,0)
    if verticals.count(0) == 0 :
        verticals.insert(0,0)
    if horizontals.count(img_hmax) == 0 :
        horizontals.insert(img_hmax,0)
    if verticals.count(img_vmax) == 0 :
        verticals.insert(img_vmax,0)

    bounding_boxes = list()

    min_h = get_minimum_separation(horizontals)
    min_v = get_minimum_separation(verticals)

    for n in range(len(verticals)-1) : #outer summation
        for i in range(len(horizontals)-1) : #inner summation
            if (horizontals[i+1] - horizontals[i]) >= min_h and (verticals[n+1] - verticals[n]) >= min_v :
                bounding_boxes.append([horizontals[i],verticals[n],horizontals[i+1],verticals[n+1]])

    return bounding_boxes

def generate_box_content(image,boxes) :
    all_boxes = list()
    for box in boxes :
        print('Creating geometry (%i)'%(len(all_boxes)))
        left = box[0]; top = box[1]; right = box[2]; bottom = box[3]
        box_arr = np.empty((right-left,bottom-top))
        for r in range(image.shape[0]) :
            for c in range(image.shape[1]) :
                if(r >= left and r < right and c >= top and c < bottom) :
                    box_arr[r-left,c-top] = image[r,c]
        all_boxes.append(box_arr)
    return all_boxes

def get_minimum_separation(arr) :
    minimum = 0
    for i in range(len(arr)-1) :
        if(arr[i+1]-arr[i] > minimum) :
            minimum = arr[i+1]-arr[i]
    return minimum/2

def plot_image_from_list(l) :
    num = len(l)
    width = 0
    height = 0
    for n in range(num) :
        width += l[n].shape[0]
        if l[n].shape[1] > height :
            height = l[n].shape[1]
    arr = np.empty((height,width))

    h=0; w=0
    for i in l :
        for r in range(i.shape[0]) :
            for c in range(i.shape[1]) :
                arr[w,h] = i[r,c]
                w+=1
            h+=1
    plot.imshow(arr)

def contrast(imge,pivot) :
    for r in range(imge.shape[0]) :
        for c in range(imge.shape[1]) :
            imge[r,c] -= (pivot - imge[r,c])
            if imge[r,c] > 255 :
                imge[r,c] = 255
            if imge[r,c] < 0 :
                imge[r,c] = 0
    return imge

def trim_bkg(arr,bkgval,tol) :
    try :
        hdim = np.trim_zeros(arr[0,:])
    except :
        hdim = arr[0,:]
    try :
        vdim = np.trim_zeros(arr[:,0])
    except :
        vdim = arr[:,0]
    hdim = np.reshape(hdim,(hdim.shape[0],1))
    vdim = np.reshape(vdim,(1,vdim.shape[0]))
    #arr = np.dot(hdim,vdim)
    #arr /= 255
    arr = arr[:,~np.all((abs(arr - bkgval) < tol), axis=0)]
    return arr[~np.all((abs(arr - bkgval) < tol), axis=1)]

def get_nn_data(imglist) :
    rimgs = list()
    for i in imglist :
        rimgs.append(rescale_img(i,(20,20)))
    return rimgs

def superimpose(base,top,pred) :
    #base[0:400,0:600] = 100
    #base[top[0]:top[2],int(top[1]*2.2)+20:int(top[3]*2.2)+20] = 200
    plot.text(top[3],(top[0]*2.9)+20,pred,fontdict=font)
    return base
#############

def subdivide_image(path, scale) :
    import random
    img = create_scaled_image(path, scale)
    bkgval = get_background_value(img)

    #higher tolerance means more cut off
    tolerance = 200#int(auto_tolerance(img))
    print(tolerance)
    img = trim_bkg(img,bkgval,tolerance)
    hz = get_horizontal_splits(img, bkgval, tolerance)
    hz.sort()
    vt = get_vertical_splits(img, bkgval,tolerance)
    vt.sort()
    box_bounds = get_bounding_boxes(hz, vt, img)
    boxes = generate_box_content(img,box_bounds)
    print(get_minimum_separation(hz), get_minimum_separation(vt))
    subs = list()
    subox_bounds = list()
    vts = list()
    print(box_bounds)
    ##
    for n in range(len(boxes)) :
        box = boxes[n]
        box = trim_bkg(box,bkgval,1)
        print(box)
        h = list((0,box.shape[0]))
        v = get_vertical_splits(box,bkgval,tolerance)
        v.sort()
        print(v)
        vts.append(v)
        ibound = get_bounding_boxes(h, v, box)
        subox_bounds.append(ibound)
        subox = generate_box_content(box,ibound)
        print(subox_bounds)
        #tolerance = random.randint(int(tolerance/4),int(tolerance/2))
        for i in range(len(subox)) :
            subox[i] = trim_bkg(contrast(subox[i],200),bkgval,tolerance)

        subs.append(subox)
    ##
    import itertools
    subs = list(itertools.chain.from_iterable(subs))
    #subox_bounds = list(itertools.chain.from_iterable(subox_bounds))
    return subs, img, subox_bounds, box_bounds, vts, vt, hz

############################################################################
import itertools
geoms, img, sbounds, bbounds, vts, vt, hz = subdivide_image("imtext.jpg",(526,446))
print(vts[0])
nnimgs = get_nn_data(geoms)
fig = plot.figure(frameon=False)
ax = plot.Axes(fig, [0., 0., 1., 1.])
ax.set_axis_off()
fig.add_axes(ax)

model = load_model_from_JSON("./uci-fonts/model[best].json","./uci-fonts/uciweights[0.99964].h5")
'''
image = nnimgs[0]
plot.imshow(image,cmap='Blues')
#fig.savefig("timgz.png", bbox_inches='tight', pad_inches=0)
image = invert(image)
image /= 255
p = predict_class_arr(image,model)
#x = predict_class_arr(image,model)
#print(x)
'''
num = len(nnimgs)
prs = list()
for g in range(num) :
    image = nnimgs[g]
    image = invert(image)
    image /= 255
    p = predict_class_arr(image,model)
    prs.append(p)
#assign letter to bounding box
offs = 0
newbounds = sbounds

y_disp = int(get_minimum_separation(vts[0]))

for s in range(1,len(newbounds)) :
    for f in range(len(newbounds[s])) :
        newbounds[s][f][0] = y_disp*s
        newbounds[s][f][2] = y_disp*s

newbounds = list(itertools.chain.from_iterable(newbounds))

font = {'family': 'sans serif',
        'color':  'red',
        'weight': 'bold',
        'size': 22,
        }

for i in range(num) :
    '''
    print(sbounds[i])
    if(i > 0) :
        diff = int((sbounds[i][1]-sbounds[i-1][3]))
        print(diff)
        for w in range(num) :
            sbounds[w][1] += diff; sbounds[w][3] += diff
        newbounds[i] = sbounds[i]
    '''
    img = superimpose(img,newbounds[i],prs[i][0])
plot.imshow(img,cmap='bone')

sentence = list(itertools.chain.from_iterable(prs))
print(''.join(sentence))
#print(hz)
#print(get_minimum_separation(hz))
#print(get_minimum_separation(vt))
```

Ryan Taylor   January 23, 2020

CFID Dataset-Based CNN:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jan  7 12:56:10 2020

@author: ryan
"""

import pandas as pd
import numpy as np
from matplotlib import pyplot as plot
import sys
import json
import pickle

split_prop = 0.85

def create_images_from_csv(csv) :
    file = pd.io.parsers.read_csv(csv)

    imgs = list()
    labels = list()

    total = len(file['font'])

    for i in range(total) :  #len(file['font'])
        if(file['m_label'][i] < 127 and file['strength'][i] == 0.4 and file['italic'][i] == 0) : #check that character is a-z,
A-Z, 0-9, special chars
            image = np.empty((20,20))
            sys.stdout.write("\rCreating image %i, %s complete" % (i, str(round((i/total) * 100))))
            sys.stdout.flush()
            for r in range(20) :
                for c in range(20) :
                    val = file["r%sc%d"%(r,c)][i]
                    image.itemset((r,c), val)
            labels.append(file['m_label'][i])
            imgs.append(image)
    return imgs, labels

def plot_multiple_images(lis, labs) :
    image = np.empty((20, 20*len(lis)))
    string = ""
    for c in range(len(lis)) :
        newimg = lis[c]
        string += str(labs[c])
        for x in range(20) :
            for y in range(20) :
                image.itemset((x,(c*20) + y), newimg.item((x,y,0)))
    plot.xlabel(string)
    plot.imshow(image[:,:])

def load_model_from_JSON(modelpath, weightpath) :
    from keras.models import model_from_json
    json_file = open(modelpath, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights(weightpath)
    return loaded_model

def pickle_data(data, path) :
    file = open(path, 'wb')
    pickle.dump(data, file)

def unpickle_data(path) :
    file = open(path, 'rb')
    return pickle.load(file)

def create_all_data() :
    file = pd.io.parsers.read_csv("./uci-fonts/fontpaths.csv")
    total = len(file['Filename'])

    images = list()
    labels = list()

    for i in range(total) :  #len(file['font'])
        ims, labs = create_images_from_csv("./uci-fonts/" + file['Filename'][i])
        images.append(ims)
        labels.append(labs)
    pickle_data(images, "./uci-fonts/allimgs2.p")
    pickle_data(labels, "./uci-fonts/alllabs2.p")
    return images, labels


def create_data() :
    imgs, labs = create_images_from_csv("./uci-fonts/ARIAL.csv")
    pickle_data(imgs, "./uci-fonts/imgs.p")
    pickle_data(labs, "./uci-fonts/labs.p")

def save_model_as_JSON(model, savepath) :
    json_string = model.to_json()
    save = open(savepath, 'w')
    save.write(json_string)
    save.close()

def load_model_from_JSON(modelpath, weightpath) :
    from keras.models import model_from_json
    json_file = open(modelpath, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights(weightpath)
    return loaded_model

def save_model_weights(model, weightpath) :
    model.save_weights(weightpath)

def grab_data(impath, lbpath) :
    import keras
    import itertools
    imgs = unpickle_data(impath)
    labs = unpickle_data(lbpath)
    a = itertools.chain.from_iterable(imgs)
    b = itertools.chain.from_iterable(labs)
    imgs = list(a)
    labs = list(b)
    print(np.asarray(imgs).shape)
    #print(len(imgs))
    X_train = np.asarray(imgs[0:round(len(imgs)*split_prop)])
    Y_train = np.asarray(labs[0:round(len(labs)*split_prop)])
    X_test = np.asarray(imgs[round(len(imgs)*split_prop):])
    Y_test = np.asarray(labs[round(len(labs)*split_prop):])
    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')
    X_train /= 255
    X_test /= 255
    Y_train -= 33
    Y_test -= 33
    classes = len(np.unique(Y_train)) + 1
    X_train = X_train.reshape((X_train.shape[0],20,20,1))
    X_test = X_test.reshape((X_test.shape[0],20,20,1))
    return X_train, Y_train, X_test, Y_test

def grab_data_and_run() :
    X_train, Y_train, X_test, Y_test = grab_data()
    mod = load_model_from_JSON("./uci-fonts/model.json", "./uci-fonts/uciweights.h5")
    model, hist = train_network(mod, X_train, Y_train, X_test, Y_test, 50, 10)
    return model, hist

def grab_data_and_create_model() :
    imgs = unpickle_data("./uci-fonts/imgs.p")
    labs = unpickle_data("./uci-fonts/labs.p")
    X_train = np.asarray(imgs[0:round(len(imgs)*split_prop)])
    Y_train = np.asarray(labs[0:round(len(labs)*split_prop)])
    X_test = np.asarray(imgs[round(len(imgs)*split_prop):])
    Y_test = np.asarray(labs[round(len(labs)*split_prop):])
    model = create_model()
    model.train_network(model, X_train, Y_train, X_test, Y_test, 10, 10)

def train_network(model, X_train, Y_train, X_test, Y_test, batch, num_epochs) :
    import keras
    Y_train = keras.utils.to_categorical(Y_train, num_classes=94)
    Y_test = keras.utils.to_categorical(Y_test, num_classes=94)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    hist = model.fit(X_train, Y_train, epochs=num_epochs, batch_size=batch,
callbacks=[keras.callbacks.callbacks.ModelCheckpoint("./uci-fonts/uciweights.h5", monitor='accuracy', verbose=1,
save_best_only=True, save_weights_only=True, mode='auto', period=1)])
    save_model_weights(model, "./uci-fonts/uciweights.h5")
    return model, hist
```

```
def create_model() :
    import keras
    from keras.layers import Conv2D, MaxPooling2D
    from keras.layers import Dense, BatchNormalization, Flatten, Dropout
    from keras.models import Sequential
    from keras.preprocessing.image import ImageDataGenerator
    import tensorflow as tf

    classes = 94#len(np.unique(Y_train)) + 1

    model = Sequential()
    model.add(Conv2D(32, (2,2), padding='same', name='Conv1', input_shape=(20,20,1), activation='tanh'))
    model.add(BatchNormalization())
    model.add(Conv2D(32, (2,2), padding='same', name='Conv2', activation='tanh'))
    model.add(BatchNormalization())
    #model.add(Conv2D(64, (2,2), padding='same', name='Conv3', input_shape=(20,20,1), activation='tanh'))
    #model.add(BatchNormalization())
    #model.add(Conv2D(64, (2,2), padding='same', name='Conv4', activation='tanh'))
    #model.add(BatchNormalization())
    model.add(Flatten())
    model.add(Dense(units=512, activation='softmax', name='Dense1', input_shape=(400,)))
    model.add(Dropout(0.25))
    model.add(Dense(units=classes, activation='softmax', name='Dense2', input_shape=(400,)))

    save_model_as_JSON(model, "./uci-fonts/model.json")
    save_model_weights(model, "./uci-fonts/weights.h5")
    return model

def get_ascii_from_predict(xin) :
    xin = np.asarray(xin, dtype='float32')
    out = list()
    for i in range(len(xin)) :
        out.append(chr(33+np.argmax(xin[i])))
        #out.append(chr(33+np.nonzero(p[i])[0]))
    return out

def predict_class(img, model) :
    p = model.predict(img, batch_size=1, verbose=1)
    p = np.asarray(p, dtype='float32')
    out = list()
    for i in range(len(p)) :
        out.append(chr(33+np.argmax(p[i])))
        #out.append(chr(33+np.nonzero(p[i])[0]))
    return out

def get_num_correct(preds, ypred) :
    num = 0
    for i in range(len(preds)) :
        if preds[i] == chr(ypred[i]+33) :
            num += 1
    return num

def predict_and_plot(xtrain, model) :
    predicts = predict_class(xtrain, model)
    #print(predicts)
    predicts = np.asarray(predicts)
    plot_multiple_images(xtrain, predicts)
    return predicts

def create_scaled_image(imgpath) :
    import cv2
    from skimage.io import imread

    img = imread(imgpath, as_gray = True)
    #img = np.asarray(img, dtype='float32')
    newimg = cv2.resize(img, (20,20))
    newimg = np.reshape(newimg, (1,20,20,1))
    return newimg

def get_single_prediction(model,index,xpred,ypred) :
    test_img = xpred[index:index + 1]
    print(chr(ypred[index]+33))
    predicts = predict_and_plot(test_img,model)
    return predicts

def get_image_prediction(imgpath, model) :
    test_img = create_scaled_image(imgpath)
    predicts = predict_and_plot(test_img,model)
    return predicts

def get_cumulative_predictions(model,endi,xpred,ypred) :
    predicts = predict_class(xpred[0:endi], model)
    print(get_num_correct(predicts, ypred))

def evaluate_model(model, xtest, ytest) :
    import keras
    ytest = keras.utils.to_categorical(ytest, num_classes=94)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    ls = model.evaluate(xtest, ytest, verbose=1)
    return ls

def unison_shuffled_copies(a, b):
    assert len(a) == len(b)
    p = np.random.permutation(len(a))
    return a[p], b[p]


#model = create_model()
    '''
imgs, labs = create_all_data()
model = create_model()
X_train, Y_train, X_predict, Y_predict = grab_data("./uci-fonts/allimgs.p", "./uci-fonts/alllabs.p")
X_train, Y_train = unison_shuffled_copies(X_train, Y_train)
X_predict, Y_predict = unison_shuffled_copies(X_predict, Y_predict)
model,hist = train_network(model, X_train, Y_train, X_predict, Y_predict, 10, 10)
#
#
#
'''
#X_train, Y_train, X_predict, Y_predict = grab_data("./uci-fonts/allimgs.p", "./uci-fonts/alllabs.p")
#X_train, Y_train = unison_shuffled_copies(X_train, Y_train)
#X_predict, Y_predict = unison_shuffled_copies(X_predict, Y_predict)
model = load_model_from_JSON("./uci-fonts/model[best].json","./uci-fonts/uciweights[0.99964].h5")
#model,hist = train_network(model, X_train, Y_train, X_predict, Y_predict, 10, 25)
#model = load_model_from_JSON("./uci-fonts/model.json","./uci-fonts/uciweights[large].h5")
#index=187106
#l = evaluate_model(model, X_predict, Y_predict)
#print(l)
i = get_image_prediction("figb.png",model)
#get_single_prediction(model,index,X_train,Y_train)
#get_cumulative_predictions(model,index,X_predict,Y_predict)
```

Ryan Taylor   January 23, 2020

Citations

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.