```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {
    "colab": {},
    "colab_type": "code",
    "collapsed": True,
    "id": "HuX3nyKJAi-T"
   },
   "outputs": [],
   "source": [
    "import numpy as np\n",
    "import pandas as pd\n",
    "import cv2\n",
    "import os\n",
    "import random\n",
    "import threading\n",
    "\n",
    "\n",
    "%matplotlib inline\n",
    "import matplotlib.pyplot as plt"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 0,
   "metadata": {
    "colab": {},
    "colab_type": "code",
   "collapsed": True,
    "id": "17wzZoqq3tZV"
   },
   "outputs": [],
   "source": [
    "from urllib.request import urlopen\n",
    "def getData(url,dirname=\"data\",img_shape=(100,100)):\n",
    "    data = 
pd.read_csv(url,sep=\"\\t\",skiprows=2,header=None,names=['Name','imagenum','url','rect','md5'
])\n",
    "    print(data.shape)\n",
    "    totalrows=data.shape[0]\n",
    "    total_personalities = data.Name.nunique()\n",
    "    current = 0\n",
    "    if not os.path.exists(dirname): os.mkdir(dirname)\n",
    "    j=0\n",
    "    for i in range(data.shape[0]):\n",
    "        if not os.path.exists(os.path.join(dirname,data.iloc[i].Name)):\n",
    "            os.mkdir(os.path.join(dirname,data.iloc[i].Name))\n",
    "            current+=1\n",
    "            print(\"{} : {}/{} {:.2f}%
done\".format(dirname,current,total_personalities,i*100/totalrows))\n",
    "            j=0\n",
    "        try:\n",
    "            resp = urlopen(data.iloc[i].url,timeout=1)\n",
    "            image = np.asarray(bytearray(resp.read()), dtype=\"uint8\")\n",
    "            image = cv2.imdecode(image, cv2.COLOR_BGR2GRAY)\n",
    "            p1,p2,p3,p4 = tuple(map(int,data.iloc[i].rect.split(',')))\n",
    "            image = image[p2:p4,p1:p3]\n",
    "            image = cv2.resize(image,img_shape,interpolation = cv2.INTER_AREA)\n",
    "            plt.imsave(os.path.join(dirname,data.iloc[i].Name,str(j)+'.jpg'),image)\n",
    "            j+=1\n",
    "        except:\n",
    "            pass"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 0,
   "metadata": {
    "colab": {},
    "colab_type": "code",
   "collapsed": True,
    "id": "xhwgqPfUKZcg"
   },
   "outputs": [],
   "source": [
    "data_e = threading.Thread(target = getData, \n",
    "                          args =
('http://www.cs.columbia.edu/CAVE/databases/pubfig/download/dev_urls.txt', 'eval'))\n",
    "data_e.start()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 0,
   "metadata": {
    "colab": {},
    "colab_type": "code",
    "collapsed": True,
    "id": "N6mvSco7IkHQ"
   },
   "outputs": [],
   "source": [
    "data_d = threading.Thread(target = getData, \n",
    "                          args =
('http://www.cs.columbia.edu/CAVE/databases/pubfig/download/eval_urls.txt', 'train'))\n",
    "data_d.start()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 0,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 1853
    },
    "colab_type": "code",
    "collapsed": True,
    "id": "G_Us_TsqsmXv",
    "outputId": "8a56f0c0-9dd7-4676-cc96-5961311fbb78"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "(16336, 5)\n",
      "(42461, 5)\n",
      "eval : 1/60 1.67% done\n",
      "train : 1/140 0.71% done\n",
      "eval : 2/60 3.33% done\n",
      "eval : 3/60 5.00% done\n",
      "train : 2/140 1.43% done\n",
      "eval : 4/60 6.67% done\n",
      "train : 3/140 2.14% done\n",
      "eval : 5/60 8.33% done\n",
      "eval : 6/60 10.00% done\n",
      "train : 4/140 2.86% done\n",
      "eval : 7/60 11.67% done\n",
"train : 5/140 3.57% done\n",
"eval : 8/60 13.33% done\n",
"train : 6/140 4.29% done\n",
"eval : 9/60 15.00% done\n",
"train : 7/140 5.00% done\n",
"eval : 10/60 16.67% done\n",
"eval : 11/60 18.33% done\n",
"eval : 12/60 20.00% done\n",
"eval : 13/60 21.67% done\n",
"train : 8/140 5.71% done\n",
"eval : 14/60 23.33% done\n",
"train : 9/140 6.43% done\n",
"eval : 15/60 25.00% done\n",
"train : 10/140 7.14% done\n",
"train : 11/140 7.86% done\n",
"eval : 16/60 26.67% done\n",
"eval : 17/60 28.33% done\n",
"eval : 18/60 30.00% done\n",
"train : 12/140 8.57% done\n",
"eval : 19/60 31.67% done\n",
"eval : 20/60 33.33% done\n",
"eval : 21/60 35.00% done\n",
"train : 13/140 9.29% done\n",
"eval : 22/60 36.67% done\n",
"train : 14/140 10.00% done\n",
"eval : 23/60 38.33% done\n",
"eval : 24/60 40.00% done\n",
"train : 15/140 10.71% done\n",
"train : 16/140 11.43% done\n",
"eval : 25/60 41.67% done\n",
"train : 17/140 12.14% done\n",
"train : 18/140 12.86% done\n",
"eval : 26/60 43.33% done\n",
"eval : 27/60 45.00% done\n",
"eval : 28/60 46.67% done\n",
"eval : 29/60 48.33% done\n",
"eval : 30/60 50.00% done\n",
"train : 19/140 13.57% done\n",
"train : 20/140 14.29% done\n",
"eval : 31/60 51.67% done\n",
"train : 21/140 15.00% done\n",
"eval : 32/60 53.33% done\n",
"eval : 33/60 55.00% done\n",
"train : 22/140 15.71% done\n",
"train : 23/140 16.43% done\n",
"eval : 34/60 56.67% done\n",
"train : 24/140 17.14% done\n",
"eval : 35/60 58.33% done\n",
"eval : 36/60 60.00% done\n",
"train : 25/140 17.86% done\n",
"eval : 37/60 61.67% done\n",
"train : 26/140 18.57% done\n",
"train : 27/140 19.29% done\n",
"train : 28/140 20.00% done\n",
"train : 29/140 20.71% done\n",
"train : 30/140 21.43% done\n",
"train : 31/140 22.14% done\n",
"eval : 38/60 63.33% done\n",
"eval : 39/60 65.00% done\n",
"train : 32/140 22.86% done\n",
"eval : 40/60 66.67% done\n",
"eval : 41/60 68.33% done\n",
"train : 33/140 23.57% done\n",
"train : 34/140 24.29% done\n",
"eval : 42/60 70.00% done\n",
"eval : 43/60 71.67% done\n",
"eval : 44/60 73.33% done\n",
"eval : 45/60 75.00% done\n",
"eval : 46/60 76.67% done\n",
"train : 35/140 25.00% done\n",
"train : 36/140 25.71% done\n",
"eval : 47/60 78.33% done\n",
"train : 37/140 26.43% done\n",
"eval : 48/60 80.00% done\n",
"train : 38/140 27.14% done\n",
"eval : 49/60 81.67% done\n",
"train : 39/140 27.86% done\n",
"eval : 50/60 83.33% done\n",
"train : 40/140 28.57% done\n",
"eval : 51/60 85.00% done\n",
"eval : 52/60 86.67% done\n",
"train : 41/140 29.29% done\n",
"train : 42/140 30.00% done\n",
"eval : 53/60 88.33% done\n",
"train : 43/140 30.71% done\n",
"eval : 54/60 90.00% done\n",
"eval : 55/60 91.67% done\n",
"train : 44/140 31.43% done\n",
      "eval : 56/60 93.33% done\n",
      "eval : 57/60 95.00% done\n",
      "train : 45/140 32.14% done\n",
      "eval : 58/60 96.67% done\n",
      "train : 46/140 32.86% done\n",
      "train : 47/140 33.57% done\n",
      "eval : 59/60 98.33% done\n"
     ]
    }
   ],
   "source": [
    "date_d.join()\n",
    "data_e.join()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {
    "colab": {},
    "colab_type": "code",
    "collapsed": True,
    "id": "xqtzET_IIkEo"
   },
   "outputs": [],
   "source": [
    "def getMiniBatch(batch_size=32,prob=0.5,path = \"train\"):\n",
    "    persons = os.listdir(path)\n",
    "    left = [];right = []\n",
    "    target = []\n",
    "    for _ in range(batch_size):\n",
    "        res = np.random.choice([0,1],p=[1-prob,prob])\n",
    "        if res==0:\n",
    "            p1,p2 = tuple(np.random.choice(persons,size=2,replace=False))\n",
    "            while len(os.listdir(os.path.join(path,p1)))<1 or
len(os.listdir(os.path.join(path,p2)))<1:\n",
    "                p1,p2 = tuple(np.random.choice(persons,size=2,replace=False))\n",
    "            p1 = os.path.join(path,p1,random.choice(os.listdir(os.path.join(path,p1))))\n",
    "            p2 = os.path.join(path,p2,random.choice(os.listdir(os.path.join(path,p2))))\n",
    "            p1,p2 =
np.expand_dims(cv2.imread(p1,0),-1),np.expand_dims(cv2.imread(p2,0),-1)\n",
    "            left.append(p1);right.append(p2)\n",
    "            target.append(0)\n",
    "        else:\n",
    "            p = np.random.choice(persons)\n",
    "            while len(os.listdir(os.path.join(path,p)))<2:\n",
    "                p = np.random.choice(persons)\n",
    "            p1,p2 = tuple(np.random.choice( os.listdir(os.path.join(path,p)), size=2,
replace=False ))\n",
    "            p1,p2 = os.path.join(path,p,p1),os.path.join(path,p,p2)\n",
    "            p1,p2 =
np.expand_dims(cv2.imread(p1,0),-1),np.expand_dims(cv2.imread(p2,0),-1)\n",
    "            left.append(p1);right.append(p2)\n",
    "            target.append(1)\n",
    "    return [np.array(left),np.array(right)],np.array(target)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "metadata": {
    "colab": {},
    "colab_type": "code",
    "collapsed": True,
    "id": "P1fzE1aLgOxk"
   },
   "outputs": [],
   "source": [
    "def test_oneshot(model,N,verbose=0):\n",
    "    \"\"\"Test average N way oneshot learning accuracy of a siamese neural net over k
one-shot tasks\"\"\"\n",
    "    if verbose:\n",
    "        pass\n",
    "        #print(\"Evaluating model on {} one-shot learning tasks ...\".format(N))\n",
    "    inputs, targets = getMiniBatch(N,path=\"eval\")\n",
    "    probs = model.predict(inputs)\n",
    "    output = (np.squeeze(probs)>0.5)*1\n",
    "    percent_correct = (output==targets).sum()*100/N\n",
    "    if verbose:\n",
    "        print(\"Got an average of {}% {} way one-shot learning
accuracy\".format(percent_correct,N))\n",
    "    return percent_correct"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": None,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 34
    },
    "colab_type": "code",
    "id": "4T6N7ezJEYXT",
    "outputId": "27487eb3-e441-4288-acfa-21760851c6a8"
   },
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "Using TensorFlow backend.\n"
     ]
    },
    {
     "data": {
      "text/plain": [
       "27417409"
      ]
     },
     "execution_count": 4,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "from keras.layers import Input, Conv2D, Dense, Flatten,MaxPooling2D\n",
    "from keras.layers import Lambda, Subtract\n",
    "from keras.models import Model, Sequential\n",
    "from keras.regularizers import l2\n",
    "from keras import backend as K\n",
    "from keras.optimizers import SGD,Adam\n",
    "from keras.losses import binary_crossentropy\n",
    "\n",
    "import numpy as np\n",
    "import os\n",
    "import matplotlib.pyplot as plt\n",
    "from sklearn.utils import shuffle\n",
    "\n",
    "\n",
    "def W_init(shape,name=None):\n",
    "    \"\"\"Initialize weights as in paper\"\"\"\n",
    "    values = np.random.normal(loc=0,scale=1e-2,size=shape)\n",
    "    return K.variable(values,name=name)\n",
    "\n",
    "#//TODO: figure out how to initialize layer biases in keras.\n",
    "def b_init(shape,name=None):\n",
    "    \"\"\"Initialize bias as in paper\"\"\"\n",
    "    values = np.random.normal(loc=0.5,scale=1e-2,size=shape)\n",
    "    return K.variable(values,name=name)\n",
    "\n",
    "input_shape = (100, 100, 1)\n",
    "left_input = Input(input_shape)\n",
    "right_input = Input(input_shape)\n",
    "\n",
    "#build convnet to use in each siamese 'leg'\n",
    "convnet = Sequential()\n",
    "convnet.add(Conv2D(64,(10,10),activation='relu',input_shape=input_shape,\n",
    "                   kernel_initializer=W_init,kernel_regularizer=l2(2e-4)))\n",
    "convnet.add(MaxPooling2D())\n",
    "convnet.add(Conv2D(128,(7,7),activation='relu',\n",
    "                   kernel_regularizer=l2(2e-4),kernel_initializer=W_init,bias_initializer=b_init))\n",
    "convnet.add(MaxPooling2D())\n",
    "convnet.add(Conv2D(128,(4,4),activation='relu',kernel_initializer=W_init,kernel_regularizer=l2(2e-4),bias_initializer=b_init))\n",
    "convnet.add(MaxPooling2D())\n",
    "convnet.add(Conv2D(256,(4,4),activation='relu',kernel_initializer=W_init,kernel_regularizer=l2(2e-4),bias_initializer=b_init))\n",
    "convnet.add(Flatten())\n",
    "convnet.add(Dense(4096,activation=\"sigmoid\",kernel_regularizer=l2(1e-3),kernel_initializer=W_init,bias_initializer=b_init))\n",
    "\n",
    "#encode each of the two inputs into a vector with the convnet\n",
    "encoded_l = convnet(left_input)\n",
    "encoded_r = convnet(right_input)\n",
    "\n",
    "#merge two encoded inputs with the l1 distance between them\n",
    "subtracted = Subtract()( [encoded_l,encoded_r]  )\n",
    "both = Lambda(lambda x: abs(x))(subtracted)\n",
    "prediction = Dense(1,activation='sigmoid',bias_initializer=b_init)(both)\n",
    "siamese_net = Model(inputs=[left_input,right_input],outputs=prediction)\n",
    "\n",
    "#optimizer = SGD(0.0004,momentum=0.6,nesterov=True,decay=0.0003)\n",
    "\n",
    "optimizer = Adam(0.00006)\n",
    "#//TODO: get layerwise learning rates and momentum annealing scheme described in paperworking\n",
    "siamese_net.compile(loss=\"binary_crossentropy\",optimizer=optimizer)\n",
    "\n",
    "siamese_net.count_params()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 0,
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/",
     "height": 1462
    },
    "colab_type": "code",
    "id": "PZj383E9eM5W",
    "outputId": "3b8181df-dcd5-41b1-b8ab-16e221da58b4"
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "iteration 0, training loss: 3.4481633, validation loss : 3.4017541\n",
      "Got an average of 49.2% 1000 way one-shot learning accuracy\n",
      "saving\n",
      "iteration 500, training loss: 1.8093039, validation loss : 1.1579119\n",
      "Got an average of 62.7% 1000 way one-shot learning accuracy\n",
      "saving\n",
      "iteration 1000, training loss: 0.9088915, validation loss : 0.8754915\n",
      "Got an average of 66.1% 1000 way one-shot learning accuracy\n",
      "saving\n",
      "iteration 1500, training loss: 0.7204882, validation loss : 0.7068923\n",
      "Got an average of 67.6% 1000 way one-shot learning accuracy\n",
      "saving\n",
      "train : 54/140 38.57% done\n",
      "iteration 2000, training loss: 0.6333932, validation loss : 0.7930869\n",
      "Got an average of 68.8% 1000 way one-shot learning accuracy\n",
      "saving\n",
      "iteration 2500, training loss: 0.5876395, validation loss : 0.7113198\n",
      "Got an average of 68.9% 1000 way one-shot learning accuracy\n",
"saving\n",
"iteration 3000, training loss: 0.5528495, validation loss : 0.5960521\n",
"Got an average of 71.5% 1000 way one-shot learning accuracy\n",
"saving\n",
"iteration 3500, training loss: 0.5160507, validation loss : 0.6767335\n",
"Got an average of 72.3% 1000 way one-shot learning accuracy\n",
"saving\n",
"train : 55/140 39.29% done\n",
"iteration 4000, training loss: 0.4908141, validation loss : 0.8932667\n",
"train : 56/140 40.00% done\n",
"Got an average of 72.3% 1000 way one-shot learning accuracy\n",
"saving\n",
"iteration 4500, training loss: 0.4689008, validation loss : 0.6380428\n",
"Got an average of 71.0% 1000 way one-shot learning accuracy\n",
"iteration 5000, training loss: 0.4518787, validation loss : 0.7520107\n",
"Got an average of 71.7% 1000 way one-shot learning accuracy\n",
"iteration 5500, training loss: 0.4216953, validation loss : 0.5597972\n",
"Got an average of 71.4% 1000 way one-shot learning accuracy\n",
"train : 57/140 40.71% done\n",
"iteration 6000, training loss: 0.4159212, validation loss : 0.7838413\n",
"Got an average of 71.7% 1000 way one-shot learning accuracy\n",
"iteration 6500, training loss: 0.3960772, validation loss : 0.7911708\n",
"Got an average of 69.7% 1000 way one-shot learning accuracy\n",
"train : 58/140 41.43% done\n",
"iteration 7000, training loss: 0.3742494, validation loss : 0.6924660\n",
"Got an average of 68.3% 1000 way one-shot learning accuracy\n",
"iteration 7500, training loss: 0.3552814, validation loss : 1.1951503\n",
"Got an average of 69.2% 1000 way one-shot learning accuracy\n",
"train : 59/140 42.14% done\n",
"iteration 8000, training loss: 0.3493278, validation loss : 0.5743055\n",
"Got an average of 71.1% 1000 way one-shot learning accuracy\n",
"iteration 8500, training loss: 0.3296640, validation loss : 0.7671475\n",
"Got an average of 68.1% 1000 way one-shot learning accuracy\n",
"train : 60/140 42.86% done\n",
"iteration 9000, training loss: 0.3253502, validation loss : 0.6926343\n",
"Got an average of 68.6% 1000 way one-shot learning accuracy\n",
"iteration 9500, training loss: 0.3142298, validation loss : 0.6399223\n",
"Got an average of 67.9% 1000 way one-shot learning accuracy\n",
"train : 61/140 43.57% done\n",
"iteration 10000, training loss: 0.3020428, validation loss : 0.7860140\n",
"Got an average of 70.1% 1000 way one-shot learning accuracy\n",
"iteration 10500, training loss: 0.2997830, validation loss : 0.8046795\n",
"Got an average of 68.6% 1000 way one-shot learning accuracy\n",
"train : 62/140 44.29% done\n",
      "iteration 11000, training loss: 0.2956488, validation loss : 0.7913840\n",
      "Got an average of 66.3% 1000 way one-shot learning accuracy\n",
      "iteration 11500, training loss: 0.2807679, validation loss : 1.1308795\n",
      "Got an average of 68.7% 1000 way one-shot learning accuracy\n",
      "train : 63/140 45.00% done\n",
      "iteration 12000, training loss: 0.2731883, validation loss : 1.0661415\n",
      "Got an average of 67.9% 1000 way one-shot learning accuracy\n",
      "train : 64/140 45.71% done\n",
      "iteration 12500, training loss: 0.2753995, validation loss : 1.4145101\n",
      "Got an average of 65.1% 1000 way one-shot learning accuracy\n",
      "iteration 13000, training loss: 0.2628590, validation loss : 1.0642104\n",
      "Got an average of 68.3% 1000 way one-shot learning accuracy\n",
      "iteration 13500, training loss: 0.2585511, validation loss : 0.7126833\n",
      "Got an average of 67.1% 1000 way one-shot learning accuracy\n",
      "iteration 14000, training loss: 0.2497812, validation loss : 1.0986179\n",
      "Got an average of 66.6% 1000 way one-shot learning accuracy\n",
      "iteration 14500, training loss: 0.2438048, validation loss : 1.2870369\n",
      "Got an average of 65.9% 1000 way one-shot learning accuracy\n",
      "train : 65/140 46.43% done\n",
      "iteration 15000, training loss: 0.2488321, validation loss : 1.2065634\n",
      "Got an average of 64.4% 1000 way one-shot learning accuracy\n",
      "iteration 15500, training loss: 0.2470939, validation loss : 1.1077709\n",
      "Got an average of 61.5% 1000 way one-shot learning accuracy\n"
     ]
    }
   ],
   "source": [
    "evaluate_every = 7000\n",
    "loss_every = 500\n",
    "batch_size = 32\n",
    "N = 1000\n",
    "best = 0\n",
    "loss_history = []\n",
    "for i in range(0,900000):\n",
    "    (inputs,targets)= getMiniBatch(batch_size,path=\"train\")\n",
    "    loss=siamese_net.train_on_batch(inputs,targets)\n",
    "    loss_history.append(loss)\n",
    "    if i % loss_every == 0:\n",
    "        vloss = siamese_net.test_on_batch(*getMiniBatch(batch_size,path=\"eval\"))\n",
    "        print(\"iteration {}, training loss: {:.7f}, validation loss :
{:.7f}\".format(i,np.mean(loss_history),vloss))\n",
    "        loss_history.clear()\n",
    "        val_acc = test_oneshot(siamese_net,N,verbose=True)\n",
    "        if val_acc >= best:\n",
  "            print(\"saving\")\n",
  "            siamese_net.save('saved_best')\n",
  "            best=val_acc"
 ]
},
{
 "cell_type": "code",
 "execution_count": None,
 "metadata": {
  "colab": {},
  "colab_type": "code",
  "id": "dgymtPvFeMzt"
 },
 "outputs": [],
 "source": [
  "val_acc = None\n",
  "while val_acc==None: \n",
  "    try:\n",
  "        siamese_net.load_weights(\"saved_best\")\n",
  "        val_acc = test_oneshot(siamese_net,1000,verbose=True)\n",
  "        print(\"Accuracy: {}\".format(val_acc))\n",
  "    except:\n",
  "        pass"
 ]
},
{
 "cell_type": "code",
 "execution_count": None,
 "metadata": {
  "colab": {},
  "colab_type": "code",
  "id": "ng7iAbNxztgI"
 },
 "outputs": [],
 "source": [
  "#haarcascade_frontalface_default.xml is saved model for face detection\n",
  "faceCascade = cv2.CascadeClassifier(\"haarcascade_frontalface_default.xml\")\n",
  "def giveAllFaces(image,BGR_input=True,BGR_output=False):\n",
  "    \"\"\"\n",
  "      return GRAY cropped_face,x,y,w,h \n",
  "    \"\"\"\n",
  "    gray = image.copy()\n",
  "    if BGR_input:\n",
  "        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)\n",
    "    faces = faceCascade.detectMultiScale(\n",
    "        gray,\n",
    "        scaleFactor=1.3,\n",
    "        minNeighbors=3,\n",
    "        minSize=(30, 30)\n",
    "    )\n",
    "    if BGR_output:\n",
    "        for (x, y, w, h) in faces:\n",
    "            yield image[y:y+h,x:x+w,:],x,y,w,h\n",
    "    else:\n",
    "        for (x, y, w, h) in faces:\n",
    "            yield gray[y:y+h,x:x+w],x,y,w,h\n",
    "\n",
    "#to draw rectangle\n",
    "#for (_,x, y, w, h) in giveAllFaces(image):\n",
    "#    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)\n",
    "\n",
    "import math\n",
    "def test(path=\"sample/tbbt.jpg\"):\n",
    "    image = cv2.imread(path)\n",
    "    faces= [ cv2.resize(face,(100,100),interpolation = cv2.INTER_AREA) for face,_,_,_,_ in giveAllFaces(image,BGR_output=True)]\n",
    "    print(\"Total Faces Detected: {}\".format(len(faces)))\n",
    "    t = math.ceil(len(faces)/2)\n",
    "    i,one = 0,[]\n",
    "    while i<t:\n",
    "        one.append(faces[i]);i+=1\n",
    "    two = one.copy()\n",
    "    while i<len(faces):\n",
    "        two[i-t] = faces[i];i+=1\n",
    "    plt.imshow(np.vstack([np.hstack(one),np.hstack(two)]))\n",
    "\n",
    "test() #other options - got.jpg, friends.jpg"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": None,
   "metadata": {},
   "outputs": [],
   "source": [
    "def putBoxText(image,x,y,w,h,text=\"unknown\"):\n",
    "    font = cv2.FONT_HERSHEY_SIMPLEX\n",
    "    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)\n",
    "    cv2.putText(image,text, (x,y-6), font, 1, (0, 255, 0), 2, cv2.LINE_AA)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": None,
   "metadata": {
    "colab": {},
    "colab_type": "code",
    "collapsed": True,
    "id": "jb0shiq-4cmL"
   },
   "outputs": [],
   "source": [
    "def putCharacters(image,db=\"database\"):\n",
    "    dbs = os.listdir(db)\n",
    "    right = np.array([ np.expand_dims(cv2.imread(os.path.join(db,x),0),-1) for x in dbs ])\n",
    "    names = [ os.path.splitext(x)[0] for x in dbs ]\n",
    "    for face,x,y,w,h in giveAllFaces(image):\n",
    "        face = cv2.resize(face,(100,100),interpolation = cv2.INTER_AREA)\n",
    "        face = np.expand_dims(face,-1)\n",
    "        left = np.array([face for _ in range(len(dbs))])\n",
    "        probs = np.squeeze(siamese_net.predict([left,right]))\n",
    "        index = np.argmax(probs)\n",
    "        prob = probs[index]\n",
    "        name = \"Unknown\"\n",
    "        if prob>0.5:\n",
    "            name = names[index]\n",
    "        putBoxText(image,x,y,w,h,text=name+\"({:.2f})\".format(prob))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": None,
   "metadata": {},
   "outputs": [],
   "source": [
    "im = cv2.imread('myself.jpg',1)\n",
    "putCharacters(im)\n",
    "plt.imshow(im)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": None,
   "metadata": {
    "collapsed": True
   },
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "accelerator": "GPU",
  "colab": {
   "collapsed_sections": [],
   "name": "Siamese networks.ipynb",
   "provenance": [],
   "version": "0.3.2"
  },
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.6.3"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 1
}
```