Mujtaba Shahid Faizi

BSCS-5A

#131818

Lab 9 of Computer Vision

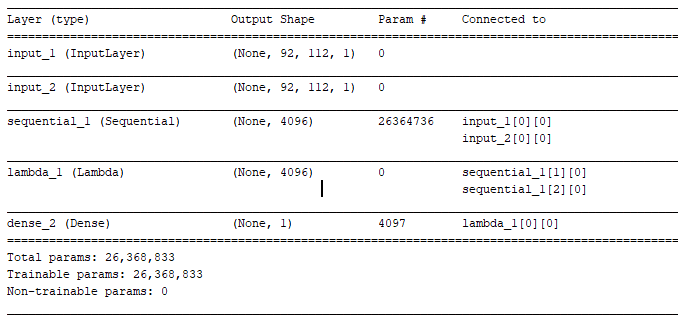
**CODE:**

**import** numpy **as** np  
**import** os  
  
**import** cv2  
**import** time  
  
**from** keras.models **import** Sequential  
**import** keras.backend **as** K  
**from** keras.utils.vis\_utils **import** plot\_model  
**from** keras.optimizers **import** Adam  
**from** keras.layers **import** Conv2D, ZeroPadding2D, Activation, Input, concatenate  
**from** keras.models **import** Model  
  
**from** keras.layers.normalization **import** BatchNormalization  
**from** keras.layers.pooling **import** MaxPooling2D  
**from** keras.layers.merge **import** Concatenate  
**from** keras.layers.core **import** Lambda, Flatten, Dense  
**from** keras.initializers **import** glorot\_uniform  
  
**from** keras.engine.topology **import** Layer  
**from** keras.regularizers **import** l2  
**from** keras **import** backend **as** K  
  
**from** sklearn.utils **import** shuffle  
  
**import** numpy.random **as** rng  
  
**def** loadimgs(path, n=0):  
 *'''  
 path => Path of train directory or test directory  
 '''* X = []  
 y = []  
 test =[]  
 test\_Y =[]  
 cat\_dict = {}  
 lang\_dict = {}  
 test\_dict = {}  
 curr\_y = n  
 curr\_test = n  
  
 *# we load every alphabet seperately so we can isolate them later* **for** alphabet **in** os.listdir(path):  
 print(**"loading alphabet: "** + alphabet)  
 lang\_dict[alphabet] = [curr\_y, **None**]  
 test\_dict[alphabet] = [curr\_y, **None**]  
 alphabet\_path = os.path.join(path, alphabet)  
 count=1  
 *# every letter/category has it's own column in the array, so load seperately* **for** letter **in** os.listdir(alphabet\_path):  
 cat\_dict[curr\_y] = (alphabet, letter)  
 category\_images = []  
 letter\_path = os.path.join(alphabet\_path, letter)  
 image = cv2.imread(letter\_path,0)  
 print(letter\_path)  
 category\_images.append(image)  
 **if** count>8:  
 test.append(curr\_y)  
 test\_Y.append(np.stack(category\_images))  
 curr\_test += 1  
 test\_dict[alphabet][1] = curr\_test - 1  
 **else**:  
 y.append(curr\_y)  
 X.append(np.stack(category\_images))  
 curr\_y += 1  
 lang\_dict[alphabet][1] = curr\_y - 1  
 count+=1  
 y = np.vstack(y)  
 X = np.stack(X)  
 test = np.stack(test)  
 test\_Y = np.vstack(test\_Y)  
 **return** X, y, lang\_dict, test, test\_Y, test\_dict  
  
Xtrain, y,train\_classes, Xval, yval,val\_classes = loadimgs(**'orl\_faces'**)  
print(Xtrain.shape,y.shape,train\_classes.keys())  
print(Xval.shape,yval.shape,val\_classes.keys())  
  
**def** initialize\_weights(shape, name=**None**):  
 *"""  
 The paper, http://www.cs.utoronto.ca/~gkoch/files/msc-thesis.pdf  
 suggests to initialize CNN layer weights with mean as 0.0 and standard deviation of 0.01  
 """* **return** np.random.normal(loc = 0.0, scale = 1e-2, size = shape)  
  
**def** initialize\_bias(shape, name=**None**):  
 *"""  
 The paper, http://www.cs.utoronto.ca/~gkoch/files/msc-thesis.pdf  
 suggests to initialize CNN layer bias with mean as 0.5 and standard deviation of 0.01  
 """* **return** np.random.normal(loc = 0.5, scale = 1e-2, size = shape)  
  
  
**def** get\_siamese\_model(input\_shape):  
 *"""  
 Model architecture based on the one provided in: http://www.cs.utoronto.ca/~gkoch/files/msc-thesis.pdf  
 """  
  
 # Define the tensors for the two input images* left\_input = Input(input\_shape)  
 right\_input = Input(input\_shape)  
  
 *# Convolutional Neural Network* model = Sequential()  
 model.add(Conv2D(64, (10, 10), activation=**'relu'**, input\_shape=input\_shape,  
 kernel\_initializer=initialize\_weights, kernel\_regularizer=l2(2e-4)))  
 model.add(MaxPooling2D())  
 model.add(Conv2D(128, (7, 7), activation=**'relu'**,  
 kernel\_initializer=initialize\_weights,  
 bias\_initializer=initialize\_bias, kernel\_regularizer=l2(2e-4)))  
 model.add(MaxPooling2D())  
 model.add(Conv2D(128, (4, 4), activation=**'relu'**, kernel\_initializer=initialize\_weights,  
 bias\_initializer=initialize\_bias, kernel\_regularizer=l2(2e-4)))  
 model.add(MaxPooling2D())  
 model.add(Conv2D(256, (4, 4), activation=**'relu'**, kernel\_initializer=initialize\_weights,  
 bias\_initializer=initialize\_bias, kernel\_regularizer=l2(2e-4)))  
 model.add(Flatten())  
 model.add(Dense(4096, activation=**'sigmoid'**,  
 kernel\_regularizer=l2(1e-3),  
 kernel\_initializer=initialize\_weights, bias\_initializer=initialize\_bias))  
  
 *# Generate the encodings (feature vectors) for the two images* encoded\_l = model(left\_input)  
 encoded\_r = model(right\_input)  
  
 *# Add a customized layer to compute the absolute difference between the encodings* L1\_layer = Lambda(**lambda** tensors: K.abs(tensors[0] - tensors[1]))  
 L1\_distance = L1\_layer([encoded\_l, encoded\_r])  
  
 *# Add a dense layer with a sigmoid unit to generate the similarity score* prediction = Dense(1, activation=**'sigmoid'**, bias\_initializer=initialize\_bias)(L1\_distance)  
  
 *# Connect the inputs with the outputs* siamese\_net = Model(inputs=[left\_input, right\_input], outputs=prediction)  
  
 *# return the model* **return** siamese\_net  
  
  
**def** get\_batch(batch\_size, s=**"train"**):  
 *"""Create batch of n pairs, half same class, half different class"""* **if** s == **'train'**:  
 X = Xtrain  
 categories = train\_classes  
 **else**:  
 X = Xval  
 categories = val\_classes  
  
 n\_classes, n\_examples, h, w = X.shape  
  
 *# randomly sample several classes to use in the batch* categories = rng.choice(n\_classes, size=(batch\_size,), replace=**False**)  
  
 *# initialize 2 empty arrays for the input image batch* pairs = [np.zeros((batch\_size, w, h, 1)) **for** i **in** range(2)]  
  
 *# initialize vector for the targets* targets = np.zeros((batch\_size,))  
  
 *# make one half of it '1's, so 2nd half of batch has same class* targets[batch\_size // 2:] = 1  
 **for** i **in** range(batch\_size):  
 category = categories[i]  
 idx\_1 = rng.randint(0, n\_examples)  
 pairs[0][i, :, :, :] = X[category, idx\_1].reshape(w, h, 1)  
 idx\_2 = rng.randint(0, n\_examples)  
  
 *# pick images of same class for 1st half, different for 2nd* **if** i >= batch\_size // 2:  
 category\_2 = category  
 **else**:  
 *# add a random number to the category modulo n classes to ensure 2nd image has a different category* category\_2 = (category + rng.randint(1, n\_classes)) % n\_classes  
  
 pairs[1][i, :, :, :] = X[category\_2, idx\_2].reshape(w, h, 1)  
  
 **return** pairs, targets  
  
**def** generate(batch\_size, s=**"train"**):  
 *"""a generator for batches, so model.fit\_generator can be used. """* **while True**:  
 pairs, targets = get\_batch(batch\_size,s)  
 **yield** (pairs, targets)  
  
  
**def** make\_oneshot\_task(N, s=**"val"**, language=**None**):  
 *"""Create pairs of test image, support set for testing N way one-shot learning. """* **if** s == **'train'**:  
 X = Xtrain  
 categories = train\_classes  
 **else**:  
 X = Xval  
 categories = val\_classes  
 n\_classes, n\_examples, w, h = X.shape  
  
 indices = rng.randint(0, n\_examples, size=(N,))  
 **if** language **is not None**: *# if language is specified, select characters for that language* low, high = categories[language]  
 **if** N > high - low:  
 **raise** ValueError(**"This language ({}) has less than {} letters"**.format(language, N))  
 categories = rng.choice(range(low, high), size=(N,), replace=**False**)  
  
 **else**: *# if no language specified just pick a bunch of random letters* categories = rng.choice(range(n\_classes), size=(N,), replace=**False**)  
 true\_category = categories[0]  
 ex1, ex2 = rng.choice(n\_examples, replace=**False**, size=(2,))  
 test\_image = np.asarray([X[true\_category, ex1, :, :]] \* N).reshape(N, w, h, 1)  
 support\_set = X[categories, indices, :, :]  
 support\_set[0, :, :] = X[true\_category, ex2]  
 support\_set = support\_set.reshape(N, w, h, 1)  
 targets = np.zeros((N,))  
 targets[0] = 1  
 targets, test\_image, support\_set = shuffle(targets, test\_image, support\_set)  
 pairs = [test\_image, support\_set]  
  
 **return** pairs, targets  
  
**def** test\_oneshot(model, N, k, s = **"val"**, verbose = 0):  
 *"""Test average N way oneshot learning accuracy of a siamese neural net over k one-shot tasks"""* n\_correct = 0  
 **if** verbose:  
 print(**"Evaluating model on {} random {} way one-shot learning tasks ... \n"**.format(k,N))  
 **for** i **in** range(k):  
 inputs, targets = make\_oneshot\_task(N,s)  
 probs = model.predict(inputs)  
 **if** np.argmax(probs) == np.argmax(targets):  
 n\_correct+=1  
 percent\_correct = (100.0 \* n\_correct / k)  
 **if** verbose:  
 print(**"Got an average of {}% {} way one-shot learning accuracy \n"**.format(percent\_correct,N))  
 **return** percent\_correct  
  
  
  
model = get\_siamese\_model((92, 112, 1))  
*# plot\_model(model,to\_file='model\_plot.png',show\_shapes=True)*model.summary()  
  
  
  
**def** euclidean\_distance\_loss(y\_true, y\_pred):  
 *"""  
 Euclidean distance loss  
 https://en.wikipedia.org/wiki/Euclidean\_distance* **:param** *y\_true: TensorFlow/Theano tensor* **:param** *y\_pred: TensorFlow/Theano tensor of the same shape as y\_true* **:return***: float  
 """* **return** K.sqrt(K.sum(K.square(y\_pred - y\_true), axis=-1))  
  
optimizer = Adam(lr = 0.00006)  
model.compile(loss=euclidean\_distance\_loss, optimizer=**'rmsprop'**)  
  
*# Hyper parameters*evaluate\_every = 200 *# interval for evaluating on one-shot tasks*batch\_size = 32  
n\_iter = 20000 *# No. of training iterations*N\_way = 20 *# how many classes for testing one-shot tasks*n\_val = 250 *# how many one-shot tasks to validate on*best = -1  
  
model\_path = **'./weights/'**print(**"Starting training process!"**)  
print(**"-------------------------------------"**)  
t\_start = time.time()  
**for** i **in** range(1, n\_iter+1):  
 (inputs,targets) = get\_batch(batch\_size)  
 loss = model.train\_on\_batch(inputs, targets)  
 **if** i % evaluate\_every == 0:  
 print(**"\n ------------- \n"**)  
 print(**"Time for {0} iterations: {1} mins"**.format(i, (time.time()-t\_start)/60.0))  
 print(**"Train Loss: {0}"**.format(loss))  
 val\_acc = test\_oneshot(model, N\_way, n\_val, verbose=**True**)  
 model.save\_weights(os.path.join(model\_path, **'weights.{}.h5'**.format(i)))  
 **if** val\_acc >= best:  
 print(**"Current best: {0}, previous best: {1}"**.format(val\_acc, best))  
 best = val\_acc

**SCREENSHOTS:**

Contrastive Loss used: Euclidian Distance function

**Network picture:**



Accuracy: 82%

