Image Similarity with Siamese Networks

Kevin Mader

last run 3 months ago · Python notebook · 1517 views
using data from Fashion MNIST · Public

Tags
- data visualization
- deep learning
- cnn
- image processing

Notebook
Overview

With the kernel I am trying to run a simple test on using Siamese networks for similarity on a slightly more complicated problem than standard MNIST. The idea is to take a randomly initialized network and apply it to images to find out how similar they are. The models should make it much easier to perform tasks like Visual Search on a database of images since it will have a simple similarity metric between 0 and 1 instead of 2D arrays.


```
In [1]:
    import numpy as np
    import os
    import pandas as pd
    from keras.preprocessing.image import ImageDataGenerator
    from keras.utils.np_utils import to_categorical
    import matplotlib.pyplot as plt

Using TensorFlow backend.
```

Load and Organize Data

Here we load and organize the data so we can easily use it inside of Keras models

```
In [2]:
    from sklearn.model_selection import train_test_split
    data_train = pd.read_csv('..//input/fashion-mnist_train.csv')
    X_full = data_train.iloc[:,1:]
    y_full = data_train.iloc[:,1]
    x_train, x_test, y_train, y_test = train_test_split(X_full, y_full, test_size = 0.3)

In [3]:
    x_train = x_train.values.reshape(-1, 28, 28, 1).astype('float32') / 255.
    x_test = x_test.values.reshape(-1, 28, 28, 1).astype('float32') / 255.
```
```python
y_train = y_train.values.astype('int')
y_test = y_test.values.astype('int')
print('Training', x_train.shape, x_train.max())
print('Testing', x_test.shape, x_test.max())
```

Training (42000, 28, 28, 1) 1.0
Testing (18000, 28, 28, 1) 1.0

```python
# reorganize by groups
train_groups = [x_train[np.where(y_train==i)[0]] for i in np.unique(y_train)]
test_groups = [x_test[np.where(y_test==i)[0]] for i in np.unique(y_train)]
print('train groups:', [x.shape[0] for x in train_groups])
print('test groups:', [x.shape[0] for x in test_groups])
```

train groups: [4165, 4155, 4162, 4196, 4258, 4246, 4239, 4184, 4230, 4165]
test groups: [1835, 1845, 1838, 1804, 1742, 1754, 1761, 1816, 1770, 1835]

### Batch Generation

Here the idea is to make usable batches for training the network. We need to create parallel inputs for the $A$ and $B$ images where the output is the distance. Here we make the naive assumption that if images are in the same group the similarity is 1 otherwise it is 0.

If we randomly selected all of the images we would likely end up with most images in different groups.

```python
def gen_random_batch(in_groups, batch_halfsize = 8):
    out_img_a, out_img_b, out_score = [], [], []
    all_groups = list(range(len(in_groups)))
    for match_group in [True, False]:
        group_idx = np.random.choice(all_groups, size = batch_halfsize)
        out_img_a += [in_groups[c_idx][np.random.choice(range(in_groups[c_idx].shape[0]))] for c_idx in group_idx]
        if match_group:
            b_group_idx = group_idx
            out_score += [1]*batch_halfsize
        else:
            # anything but the same group
            non_group_idx = [np.random.choice([i for i in all_groups if i!=c_idx]) for c_idx in group_idx]
            b_group_idx = non_group_idx
            out_score += [0]*batch_halfsize
```

Validate Data

Here we make sure the generator is doing something sensible, we show the images and their similarity percentage.

```
In [6]:
    pv_a, pv_b, pv_sim = gen_random_batch(train_groups, 3)
    fig, m_axs = plt.subplots(2, pv_a.shape[0], figsize = (12, 6))
    for c_a, c_b, c_d, (ax1, ax2) in zip(pv_a, pv_b, pv_sim, m_axs.T):
        ax1.imshow(c_a[:,:,0])
        ax1.set_title('Image A')
        ax1.axis('off')
        ax2.imshow(c_b[:,:,0])
        ax2.set_title('Image B
Similarity: %3.0f%%' % (100*c_d))
        ax2.axis('off')
```

Feature Generation

Here we make the feature generation network to process images into features. The network starts off randomly initialized and will be trained to generate useful vector features from input images (hopefully)
In [7]:
from keras.models import Model
from keras.layers import Input, Conv2D, BatchNormalization, MaxPool2D, Activation, Flatten, Dense, Dropout

img_in = Input(shape = x_train.shape[1:], name = 'FeatureNet_ImageInput')
n_layer = img_in
for i in range(2):
    n_layer = Conv2D(8*2**i, kernel_size = (3,3), activation = 'linear')(n_layer)
    n_layer = BatchNormalization()(n_layer)
    n_layer = Activation('relu')(n_layer)
    n_layer = Conv2D(16*2**i, kernel_size = (3,3), activation = 'linear')(n_layer)
    n_layer = BatchNormalization()(n_layer)
    n_layer = Activation('relu')(n_layer)
    n_layer = MaxPool2D((2,2))(n_layer)
n_layer = Flatten()(n_layer)
n_layer = Dense(32, activation = 'linear')(n_layer)
n_layer = Dropout(0.5)(n_layer)
n_layer = BatchNormalization()(n_layer)
    n_layer = Activation('relu')(n_layer)
feature_model = Model(inputs = [img_in], outputs = [n_layer], name = 'FeatureGenerationModel')
feature_model.summary()

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeatureNet_ImageInput</td>
<td>(Input (None, 28, 28, 1)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_1 (Conv2D)</td>
<td>(None, 26, 26, 8)</td>
<td>80</td>
</tr>
<tr>
<td>batch_normalization_1</td>
<td>(Batch (None, 26, 26, 8)</td>
<td>32</td>
</tr>
<tr>
<td>activation_1 (Activation)</td>
<td>(None, 26, 26, 8)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_2 (Conv2D)</td>
<td>(None, 24, 24, 16)</td>
<td>1168</td>
</tr>
<tr>
<td>batch_normalization_2</td>
<td>(Batch (None, 24, 24, 16)</td>
<td>64</td>
</tr>
<tr>
<td>activation_2 (Activation)</td>
<td>(None, 24, 24, 16)</td>
<td>0</td>
</tr>
<tr>
<td>max_pooling2d_1 (MaxPooling2)</td>
<td>(None, 12, 12, 16)</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_3 (Conv2D)</td>
<td>(None, 10, 10, 16)</td>
<td>2320</td>
</tr>
</tbody>
</table>
Siamese Model

We apply the feature generating model to both images and then combine them together to predict if they are similar or not. The model is designed to very simple. The ultimate idea is when a new image is taken that a feature vector can be calculated for it using the `FeatureGenerationModel`. All existing images have been pre-calculated and stored in a database of feature vectors. The model can be applied using a few vector additions and multiplications to determine the most similar images. These operations can be implemented as a stored procedure or similar task inside the database itself since they do not require an entire deep learning framework to run.

```
In [8]:
from keras.layers import concatenate

img_a_in = Input(shape = x_train.shape[1:], name = 'ImageA_Input')
img_b_in = Input(shape = x_train.shape[1:], name = 'ImageB_Input')
img_a_feat = feature_model(img_a_in)
img_b_feat = feature_model(img_b_in)
combined_features = concatenate([img_a_fea, img_b_fea], name = 'merge_feature')
```
combined_features = Dense(16, activation = 'linear')(combined_features)
combined_features = BatchNormalization()(combined_features)
combined_features = Dense(4, activation = 'linear')(combined_features)
combined_features = BatchNormalization()(combined_features)
combined_features = Dense(1, activation = 'sigmoid')(combined_features)
similarity_model = Model(inputs = [img_a_in, img_b_in], outputs = [combined_features], name = 'Similarity_Model')
similarity_model.summary()

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
<th>Connected to</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageA_Input (InputLayer)</td>
<td>(None, 28, 28, 1)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ImageB_Input (InputLayer)</td>
<td>(None, 28, 28, 1)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>FeatureGenerationModel (Model)</td>
<td>(None, 32)</td>
<td>25040</td>
<td>ImageA_Input[0][0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ImageB_Input[0][0]</td>
</tr>
<tr>
<td>merge_features (Concatenate)</td>
<td>(None, 64)</td>
<td>0</td>
<td>FeatureGenerationModel[1][0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FeatureGenerationModel[2][0]</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 16)</td>
<td>1040</td>
<td>merge_features[0][0]</td>
</tr>
<tr>
<td>batch_normalization_6 (BatchNor)</td>
<td>(None, 16)</td>
<td>64</td>
<td>dense_2[0][0]</td>
</tr>
</tbody>
</table>
In [9]:
# setup the optimization process
similarity_model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics = ['mae'])

Visual Model Feedback

Here we visualize what the model does by taking a small sample of randomly selected A and B images the first half from the same category and the second from different categories. We then show the actual distance (0 for the same category and 1 for different categories) as well as the model predicted distance. The first run here is with a completely untrained network so we do not expect meaningful results.

In [10]:
def show_model_output(nb_examples = 3):
    pv_a, pv_b, pv_sim = gen_random_batch(test_groups, nb_examples)
    pred_sim = similarity_model.predict([pv_a, pv_b])
    fig, m_axs = plt.subplots(2, pv_a.shape[0], figsize = (12, 6))
for c_a, c_b, c_d, p_d, (ax1, ax2) in zip(pv_a, pv_b, pv_sim, pred_sim, m_axs.T):
    ax1.imshow(c_a[:, :, 0])
    ax1.set_title('Image A
Actual: %3.0f%%' % (100*c_d))
    ax1.axis('off')
    ax2.imshow(c_b[:, :, 0])
    ax2.set_title('Image B
Predicted: %3.0f%%' % (100*p_d))
    ax2.axis('off')
return fig

# a completely untrained model
_ = show_model_output()

In [11]:

# make a generator out of the data
def siam_gen(in_groups, batch_size = 32):
    while True:
        pv_a, pv_b, pv_sim = gen_random_batch(train_groups, batch_size//2)
        yield [pv_a, pv_b], pv_sim

# we want a constant validation group to have a frame of reference for model performance
valid_a, valid_b, valid_sim = gen_random_batch(test_groups, 1024)
loss_history = similarity_model.fit_generator(siam_gen(train_groups),
                                           steps_per_epoch = 500,
                                           validation_data=([valid_a, valid_b], valid_sim),
                                           epochs = 10,
                                           verbose = True)

Epoch 1/10
In [12]:
    _ = show_model_output()
T-Shirt vs Ankle Boot-Plot

Here we take an random t-shirt and ankle boot (categories 0 and 9) images and calculate the distance using our network to the other images

```
In [13]:
t_shirt_vec = np.stack([train_groups[0][0]*x_test.shape[0], 0])
t_shirt_score = similarity_model.predict([t_shirt_vec, x_test], verbose = True, batch_size = 128)
ankle_boot_vec = np.stack([train_groups[-1][0]*x_test.shape[0], 0])
ankle_boot_score = similarity_model.predict([ankle_boot_vec, x_test], verbose = True, batch_size = 128)
```

```
18000/18000 [==============================] - 21s 1ms/step
18000/18000 [==============================] - 20s 1ms/step
```

```
In [14]:
obj_categories = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                  'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
colors = plt.cm.rainbow(np.linspace(0, 1, 10))
plt.figure(figsize=(10, 10))

for c_group, (c_color, c_label) in enumerate(zip(colors, obj_categories)):
    plt.scatter(t_shirt_score[np.where(y_test == c_group), 0],
                ankle_boot_score[np.where(y_test == c_group), 0],
                marker='.',
                color=c_color,
                linewidth='1',
                alpha=0.8,
                label=c_label)
plt.xlabel('T-Shirt Dimension')
plt.ylabel('Ankle-Boot Dimension')
plt.title('T-Shirt and Ankle-Boot Dimension')
plt.legend(loc='best')
plt.savefig('tshirt-boot-dist.png')
plt.show(block=False)
```
Examining the Features

Here we aim to answer the more general question: did we generate useful features with the Feature Generation model? And how can we visualize this.

```python
In [15]:
    x_test_features = feature_model.predict(x_test, verbose = True, batch_size=128)
```

18000/18000 [==============================] - 11s 612us/step

Neighbor Visualization

For this we use the TSNE neighborhood embedding to visualize the features on a 2D plane and see if it roughly corresponds to the groups. We use the test data for this example as well since the training has been contaminated.
In [16]:

```python
%%time
from sklearn.manifold import TSNE
tsne_obj = TSNE(n_components=2,
                init='pca',
                random_state=101,
                method='barnes_hut',
                n_iter=500,
                verbose=2)

tsne_features = tsne_obj.fit_transform(x_test_features)
```

[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 18000 samples in 0.226s...
[t-SNE] Computed neighbors for 18000 samples in 4.028s...
[t-SNE] Computed conditional probabilities for sample 1000 / 18000
[t-SNE] Computed conditional probabilities for sample 2000 / 18000
[t-SNE] Computed conditional probabilities for sample 3000 / 18000
[t-SNE] Computed conditional probabilities for sample 4000 / 18000
[t-SNE] Computed conditional probabilities for sample 5000 / 18000
[t-SNE] Computed conditional probabilities for sample 6000 / 18000
[t-SNE] Computed conditional probabilities for sample 7000 / 18000
[t-SNE] Computed conditional probabilities for sample 8000 / 18000
[t-SNE] Computed conditional probabilities for sample 9000 / 18000
[t-SNE] Computed conditional probabilities for sample 10000 / 18000
[t-SNE] Computed conditional probabilities for sample 11000 / 18000
[t-SNE] Computed conditional probabilities for sample 12000 / 18000
[t-SNE] Computed conditional probabilities for sample 13000 / 18000
[t-SNE] Computed conditional probabilities for sample 14000 / 18000
[t-SNE] Computed conditional probabilities for sample 15000 / 18000
[t-SNE] Computed conditional probabilities for sample 16000 / 18000
[t-SNE] Computed conditional probabilities for sample 17000 / 18000
[t-SNE] Computed conditional probabilities for sample 18000 / 18000
[t-SNE] Mean sigma: 0.097702
[t-SNE] Computed conditional probabilities in 1.213s
[t-SNE] Iteration 50: error = 82.1846161, gradient norm = 0.0019173 (50 iterations in 27.468s)
[t-SNE] Iteration 100: error = 80.4134293, gradient norm = 0.0010669 (50 iterations in 26.792s)
[t-SNE] Iteration 150: error = 79.5910645, gradient norm = 0.0007335 (50 iterations in 27.382s)
[t-SNE] Iteration 200: error = 79.0950394, gradient norm = 0.0005696 (50 iterations in 27.344s)
[t-SNE] Iteration 250: error = 78.7620468, gradient norm = 0.0004646 (50 iterations in 27.344s)
In [17]:

    obj_categories = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
                      'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
    colors = plt.cm.rainbow(np.linspace(0, 1, 10))
    plt.figure(figsize=(10, 10))

    for c_group, (c_color, c_label) in enumerate(zip(colors, obj_categories)):
        plt.scatter(tsne_features[np.where(y_test == c_group), 0],
                    tsne_features[np.where(y_test == c_group), 1],
                    marker='o',
                    color=c_color,
                    linewidth='1',
                    alpha=0.8,
                    label=c_label)
    plt.xlabel('Dimension 1')
    plt.ylabel('Dimension 2')
    plt.title('t-SNE on Testing Samples')
    plt.legend(loc='best')
    plt.savefig('clothes-dist.png')
    plt.show(block=False)
In [18]:

    feature_model.save('fashion_feature_model.h5')

In [19]:

    similarity_model.save('fashion_similarity_model.h5')

---------------------------------------------------------------------------
TypeError                Traceback (most recent call last)
<ipython-input-19-bdbb4006d9c0> in <module>()
     1 similarity_model.save('fashion_similarity_model.h5')

/opt/conda/lib/python3.6/site-packages/Keras-2.1.2-py3.6.egg/keras/engine/topology.py in save(self, filepath, overwrite, include_optimizer)
    2563        
    2564        from ..models import save_model
-> 2565        save_model(self, filepath, overwrite, include_optimizer)
    2566    
    2567    def save_weights(self, filepath, overwrite=True):

/opt/conda/lib/python3.6/site-packages/Keras-2.1.2-py3.6.egg/keras/models.py in save_model(model, filepath, overwrite, include_optimizer)
    145        if symbolic_weights:
    146            optimizer_weights_group = f.create_group('optimize
----> 150                  r_weights')


```python
weight_values = K.batch_get_value(symbolic_weights)
weight_names = []
for i, (w, val) in enumerate(zip(symbolic_weights, weight_values)):
```

Notebook | Code | Data | Output | Comments | Log | Versions | Forks | Fork Notebook
--- | --- | --- | --- | --- | --- | --- | --- | ---
2209 | if ops: |  |  |  |  |  |  |  
2210 | return get_session().run(ops) |  |  |  |  |  |  |  
2211 | else: |  |  |  |  |  |  |  
2212 | return [] |  |  |  |  |  |  |  

```
/opt/conda/lib/python3.6/site-packages/tensorflow/python/client/session.py in run(self, fetches, feed_dict, options, run_metadata)
 887       try:
 888       result = self._run(None, fetches, feed_dict, options_ptr,
--> 889       run_metadata_ptr)
 890       if run_metadata:
 891       proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

```

```
/opt/conda/lib/python3.6/site-packages/tensorflow/python/client/session.py in _run(self, handle, fetches, feed_dict, options, run_metadata)
 1103     # Create a fetch handler to take care of the structure of fetches.
 1104     fetch_handler = _FetchHandler(
--> 1105     self._graph, fetches, feed_dict_tensor, feed_handles=feed_handles)
 1106
 1107     # Run request and get response.

```

```
/opt/conda/lib/python3.6/site-packages/tensorflow/python/client/session.py in __init__(self, graph, fetches, feeds, feed_handles)
 412     
 413     with graph.as_default():
--> 414     self._fetch_mapper = _FetchMapper.for_fetch(fetches)
 415     self._fetches = []
 416     self._targets = []

```

```
/opt/conda/lib/python3.6/site-packages/tensorflow/python/client/session.py in for_fetch(fetch)
 232     elif isinstance(fetch, (list, tuple)):
```

Did you find this Kernel useful?
Show your appreciation with an upvote

Great job. I learned a lot from here. Have you tried other model structure for image similarity modeling? Models which is mentioned in paper "Learning to Compare Image Patches via Convolutional Neural Networks" and "Matchnet: unifying feature and metric learning for patch-based Matching".