C1_W1_Lab_3_siamese-network

December 23, 2020

1 Ungraded Lab: Implement a Siamese network

This lab will go through creating and training a multi-input model. You will build a basic Siamese Network to find the similarity or dissimilarity between items of clothing. For Week 1, you will just focus on constructing the network. You will revisit this lab in Week 2 when we talk about custom loss functions.

1.1 Imports

[1]:
```
try:
# %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass

import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Flatten, Dense, Dropout, Lambda
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.python.keras.utils.vis_utils import plot_model
from tensorflow.keras import backend as K

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageFont, ImageDraw
import random
```

1.2 Prepare the Dataset

First define a few utilities for preparing and visualizing your dataset.

[2]:
```
def create_pairs(x, digit_indices):
    '''Positive and negative pair creation.
    Alternates between positive and negative pairs.
```

```
    '''
    pairs = []
    labels = []
    n = min([len(digit_indices[d]) for d in range(10)]) - 1

    for d in range(10):
        for i in range(n):
            z1, z2 = digit_indices[d][i], digit_indices[d][i + 1]
            pairs += [[x[z1], x[z2]]]
            inc = random.randrange(1, 10)
            dn = (d + inc) % 10
            z1, z2 = digit_indices[d][i], digit_indices[dn][i]
            pairs += [[x[z1], x[z2]]]
            labels += [1, 0]

    return np.array(pairs), np.array(labels)


def create_pairs_on_set(images, labels):

    digit_indices = [np.where(labels == i)[0] for i in range(10)]
    pairs, y = create_pairs(images, digit_indices)
    y = y.astype('float32')

    return pairs, y


def show_image(image):
    plt.figure()
    plt.imshow(image)
    plt.colorbar()
    plt.grid(False)
    plt.show()
```

You can now download and prepare our train and test sets. You will also create pairs of images that will go into the multi-input model.

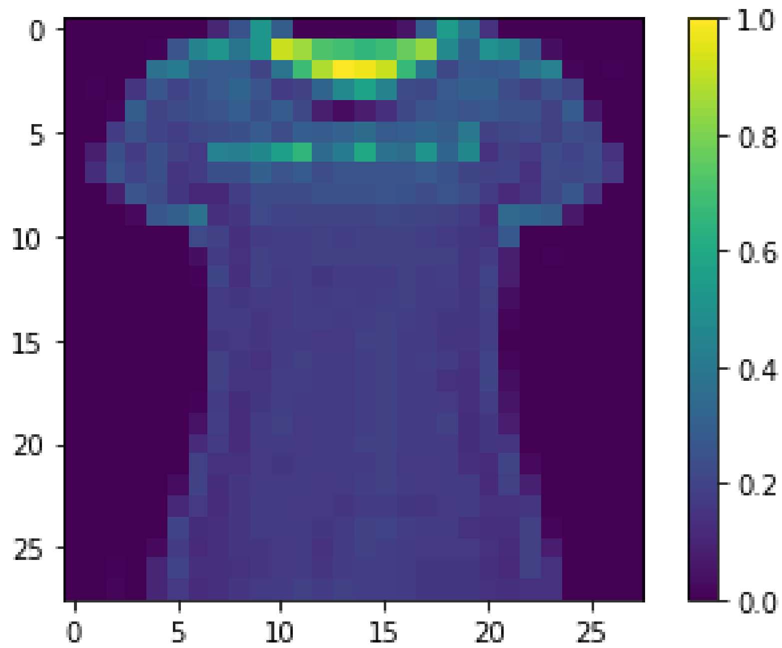
[3]:
```
# load the dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.
 ↪load_data()

# prepare train and test sets
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')

# normalize values
train_images = train_images / 255.0
```

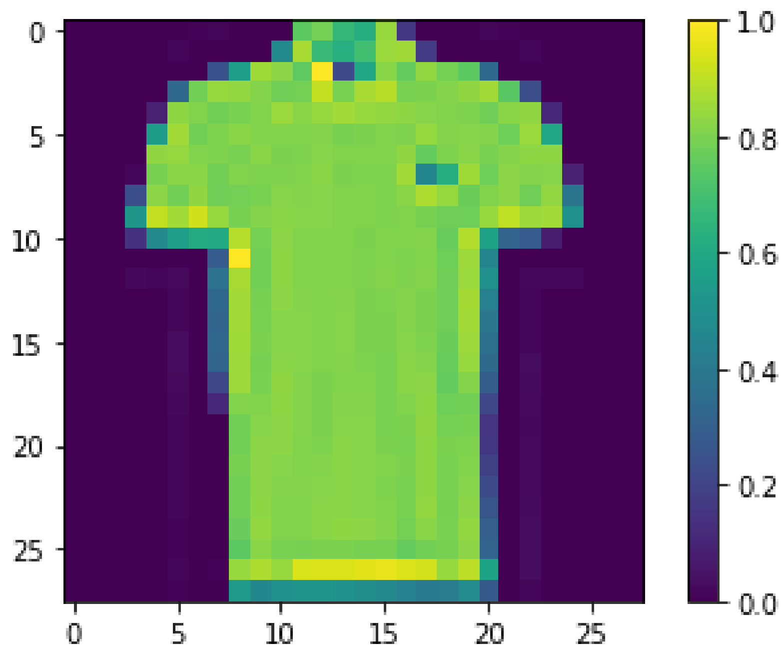
```
test_images = test_images / 255.0

# create pairs on train and test sets
tr_pairs, tr_y = create_pairs_on_set(train_images, train_labels)
ts_pairs, ts_y = create_pairs_on_set(test_images, test_labels)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
32768/29515 [==================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [==============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

You can see a sample pair of images below.

[4]:
```
# array index
this_pair = 8

# show images at this index
show_image(ts_pairs[this_pair][0])
show_image(ts_pairs[this_pair][1])

# print the label for this pair
print(tr_y[this_pair])
```

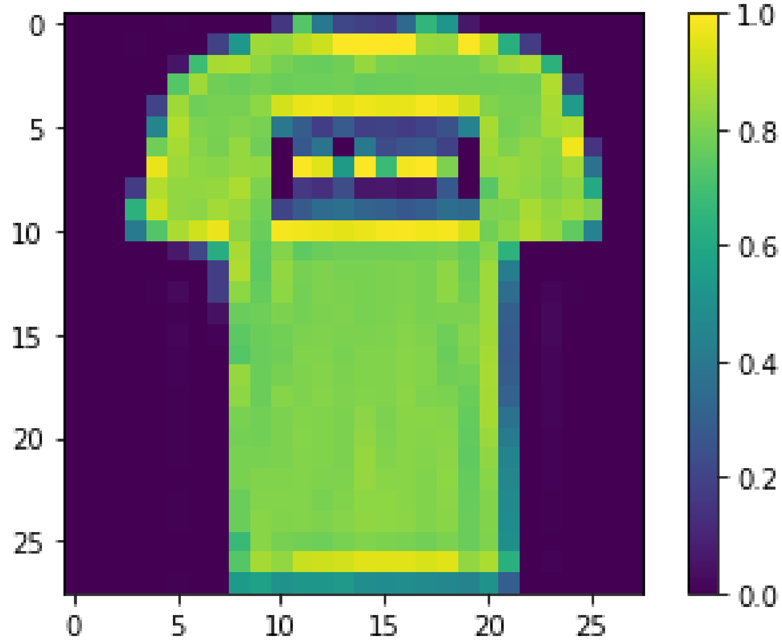


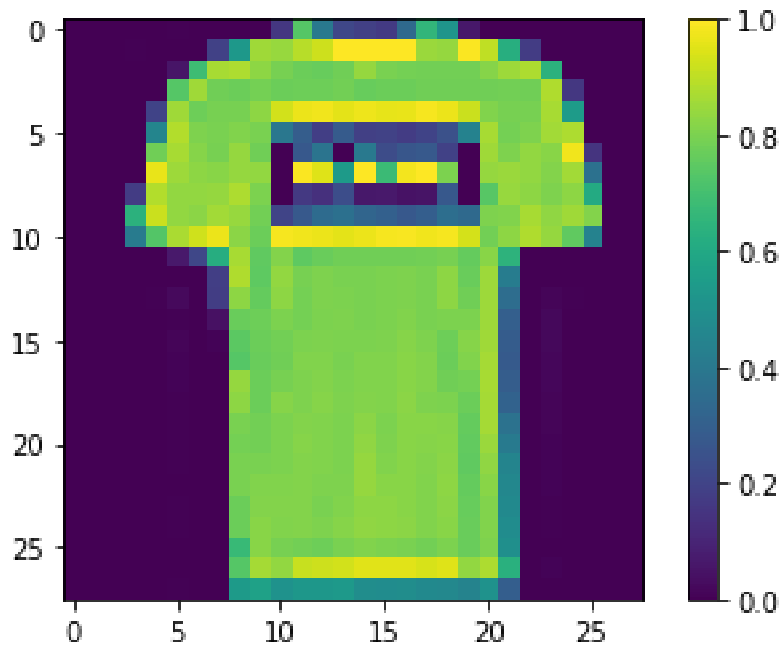


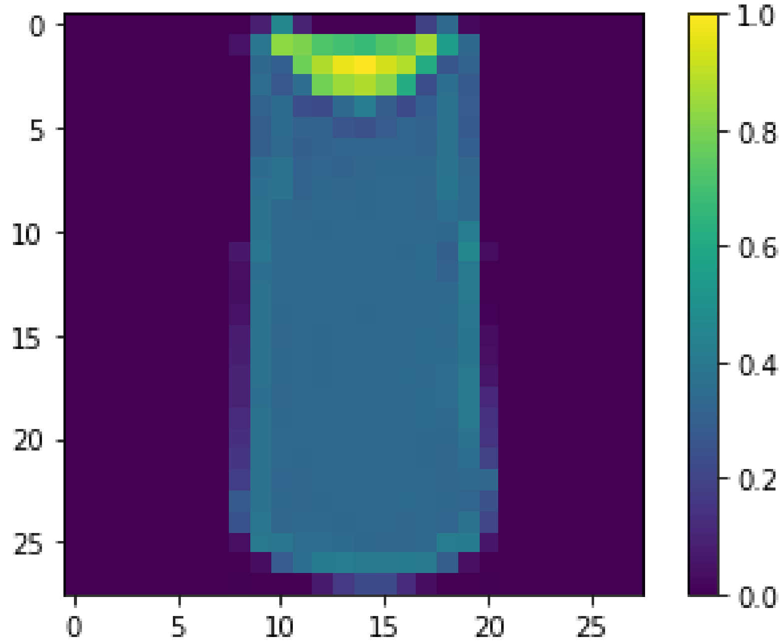

```
1.0
```

[5]:
```
# print other pairs
```

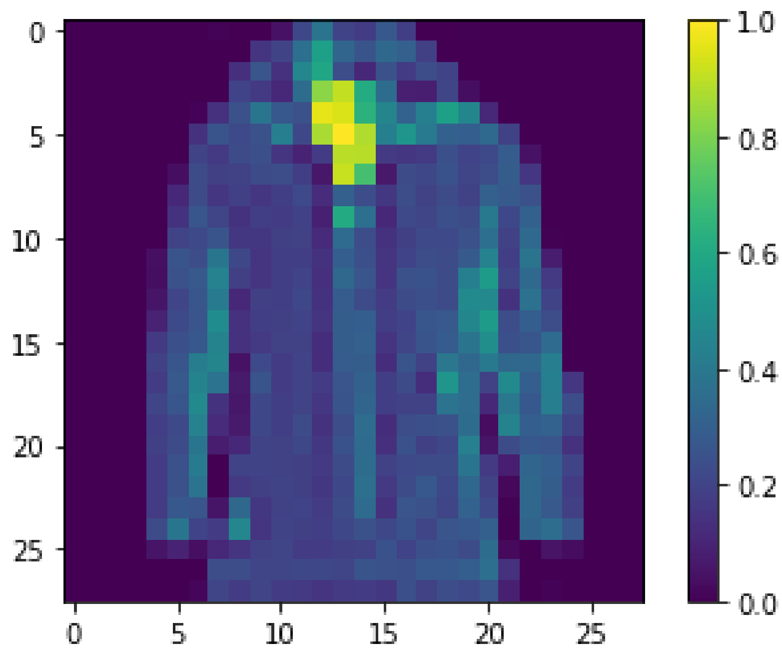
```
show_image(tr_pairs[:,0][0])
show_image(tr_pairs[:,0][1])

show_image(tr_pairs[:,1][0])
show_image(tr_pairs[:,1][1])
```







0
5
10
15
20
25
0
5
10
15
20
25
1.0
0.8
0.6
0.4
0.2
0.0



0
5
10
15
20
25
0
5
10
15
20
25
1.0
0.8
0.6
0.4
0.2
0.0

1.3 Build the Model

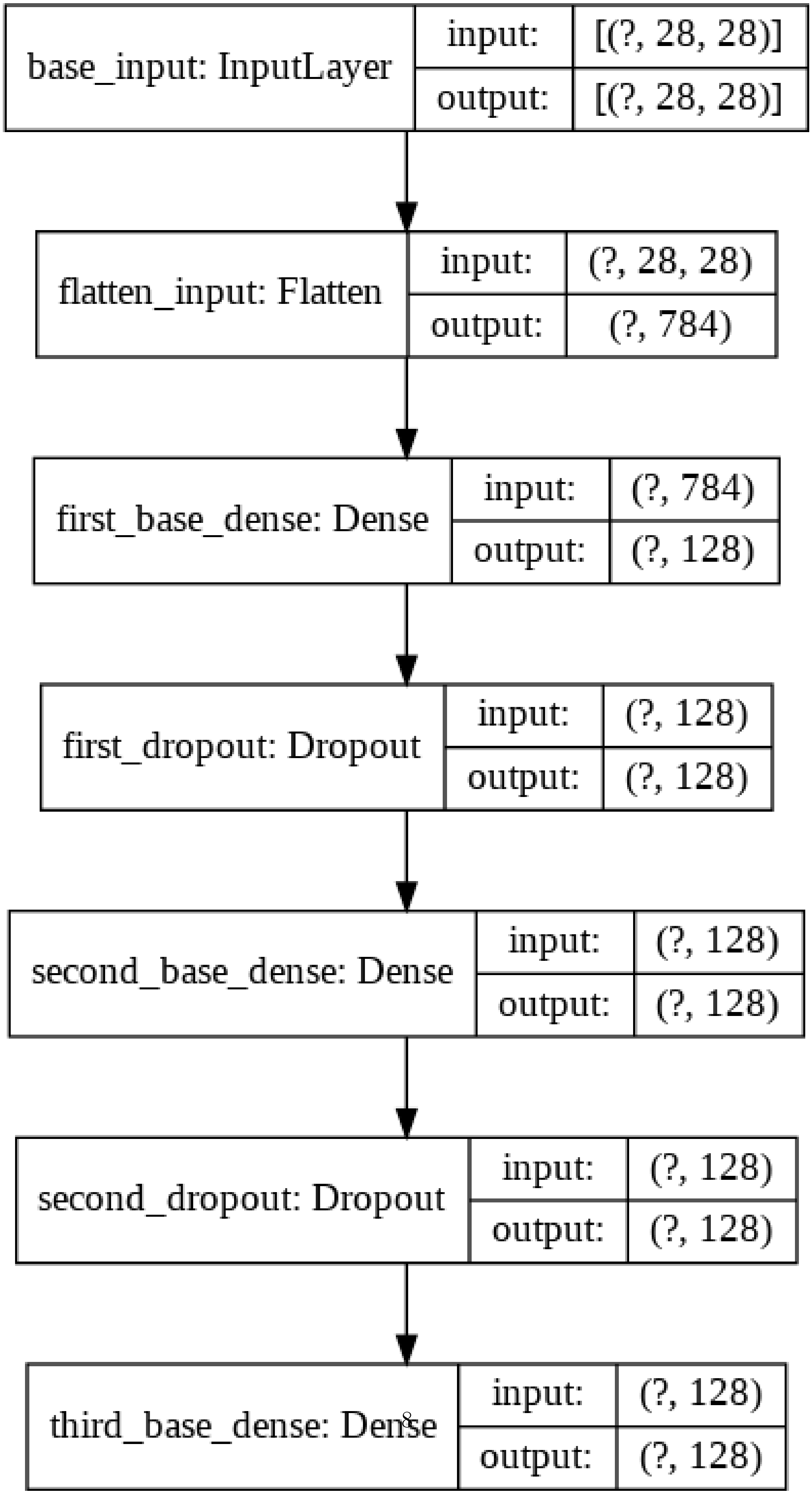
Next, you'll define some utilities for building our model.

[6]:
```
def initialize_base_network():
    input = Input(shape=(28,28,), name="base_input")
    x = Flatten(name="flatten_input")(input)
    x = Dense(128, activation='relu', name="first_base_dense")(x)
    x = Dropout(0.1, name="first_dropout")(x)
    x = Dense(128, activation='relu', name="second_base_dense")(x)
    x = Dropout(0.1, name="second_dropout")(x)
    x = Dense(128, activation='relu', name="third_base_dense")(x)

    return Model(inputs=input, outputs=x)


def euclidean_distance(vects):
    x, y = vects
    sum_square = K.sum(K.square(x - y), axis=1, keepdims=True)
    return K.sqrt(K.maximum(sum_square, K.epsilon()))


def eucl_dist_output_shape(shapes):
    shape1, shape2 = shapes
    return (shape1[0], 1)
```

Let's see how our base network looks. This is where the two inputs will pass through to generate an output vector.

[7]:
```
base_network = initialize_base_network()
plot_model(base_network, show_shapes=True, show_layer_names=True,␣
 ↪to_file='base-model.png')
```

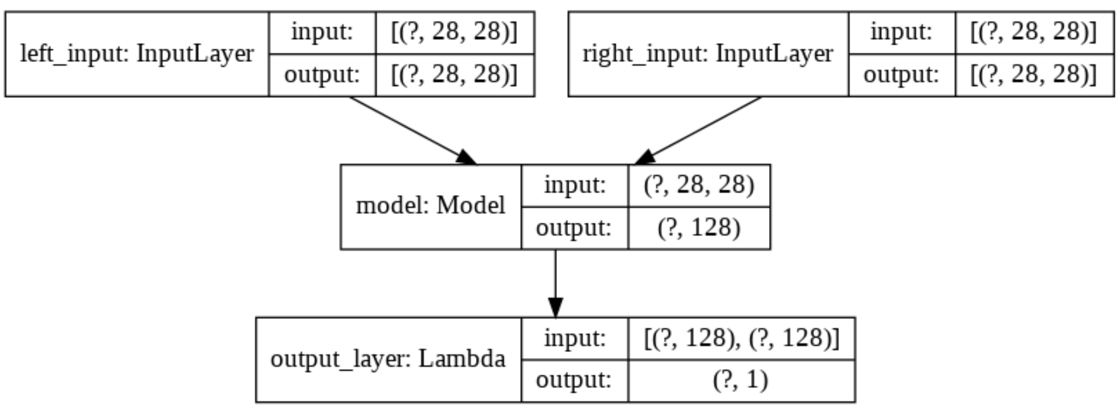
[7]:


base_input: InputLayer
input:
[(?, 28, 28)]
output:
[(?, 28, 28)]
flatten_input: Flatten
input:
(?, 28, 28)
output:
(?, 784)
first_base_dense: Dense
input:
(?, 784)
output:
(?, 128)
first_dropout: Dropout
input:
(?, 128)
output:
(?, 128)
second_base_dense: Dense
input:
(?, 128)
output:
(?, 128)
second_dropout: Dropout
input:
(?, 128)
output:
(?, 128)
third_base_dense: Dense
input:
(?, 128)
output:
(?, 128)

Let's now build the Siamese network. The plot will show two inputs going to the base network.

[8]:
```
# create the left input and point to the base network
input_a = Input(shape=(28,28,), name="left_input")
vect_output_a = base_network(input_a)

# create the right input and point to the base network
input_b = Input(shape=(28,28,), name="right_input")
vect_output_b = base_network(input_b)

# measure the similarity of the two vector outputs
output = Lambda(euclidean_distance, name="output_layer",␣
↪output_shape=eucl_dist_output_shape)([vect_output_a, vect_output_b])

# specify the inputs and output of the model
model = Model([input_a, input_b], output)

# plot model graph
plot_model(model, show_shapes=True, show_layer_names=True, to_file='outer-model.
↪png')
```

[8]:



1.4 Train the Model

You can now define the custom loss for our network and start training. Don't worry about why it's written as a nested function just yet. You will revisit this in Week 2.

[9]:
```
def contrastive_loss_with_margin(margin):
    def contrastive_loss(y_true, y_pred):
```

```
        '''Contrastive loss from Hadsell-et-al.'06
        http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf
        '''
        square_pred = K.square(y_pred)
        margin_square = K.square(K.maximum(margin - y_pred, 0))
        return K.mean(y_true * square_pred + (1 - y_true) * margin_square)
    return contrastive_loss
```

[10]:
```
rms = RMSprop()
model.compile(loss=contrastive_loss_with_margin(margin=1), optimizer=rms)
history = model.fit([tr_pairs[:,0], tr_pairs[:,1]], tr_y, epochs=20,␣
 ↪batch_size=128, validation_data=([ts_pairs[:,0], ts_pairs[:,1]], ts_y))
```

```
Train on 119980 samples, validate on 19980 samples
Epoch 1/20
119980/119980 [==============================] - 8s 69us/sample - loss: 0.1114 -
val_loss: 0.0863
Epoch 2/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0796 -
val_loss: 0.0735
Epoch 3/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0714 -
val_loss: 0.0714
Epoch 4/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0664 -
val_loss: 0.0732
Epoch 5/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0637 -
val_loss: 0.0647
Epoch 6/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0612 -
val_loss: 0.0675
Epoch 7/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0598 -
val_loss: 0.0641
Epoch 8/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0585 -
val_loss: 0.0654
Epoch 9/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0574 -
val_loss: 0.0632
Epoch 10/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0563 -
val_loss: 0.0636
Epoch 11/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0555 -
val_loss: 0.0670
```

```
Epoch 12/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0549 -
val_loss: 0.0657
Epoch 13/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0539 -
val_loss: 0.0627
Epoch 14/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0536 -
val_loss: 0.0609
Epoch 15/20
119980/119980 [==============================] - 8s 65us/sample - loss: 0.0533 -
val_loss: 0.0626
Epoch 16/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0528 -
val_loss: 0.0622
Epoch 17/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0529 -
val_loss: 0.0623
Epoch 18/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0517 -
val_loss: 0.0619
Epoch 19/20
119980/119980 [==============================] - 8s 64us/sample - loss: 0.0512 -
val_loss: 0.0615
Epoch 20/20
119980/119980 [==============================] - 8s 63us/sample - loss: 0.0506 -
val_loss: 0.0621
```

1.5 Model Evaluation

As usual, you can evaluate our model by computing the accuracy and observing the metrics during training.

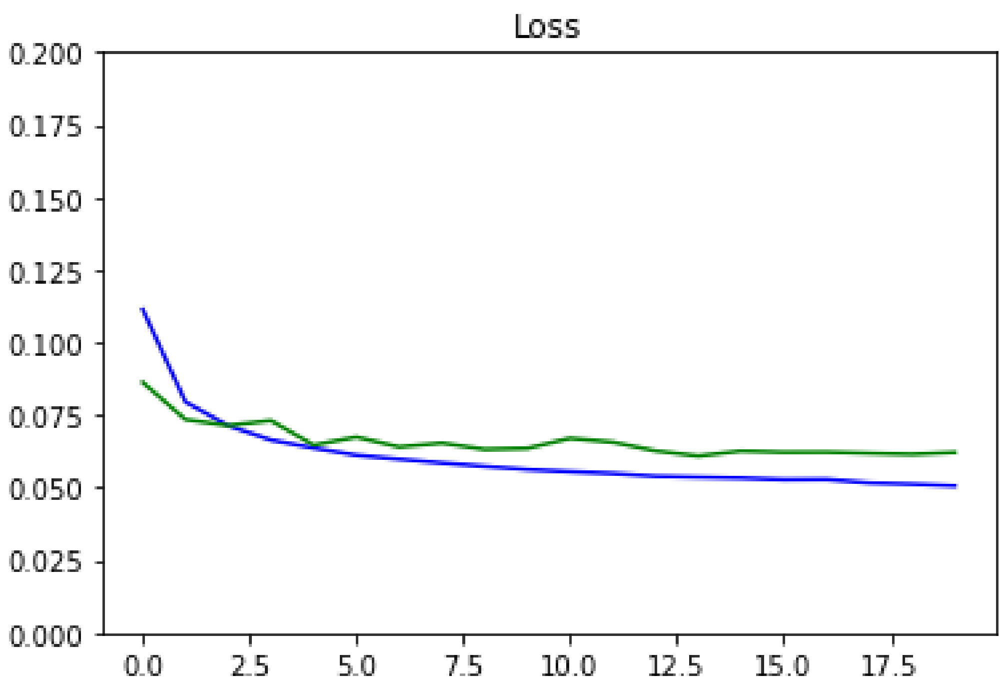
[11]:
```
def compute_accuracy(y_true, y_pred):
    '''Compute classification accuracy with a fixed threshold on distances.
    '''
    pred = y_pred.ravel() > 0.5
    return np.mean(pred == y_true)
```

[12]:
```
loss = model.evaluate(x=[ts_pairs[:,0],ts_pairs[:,1]], y=ts_y)

y_pred_train = model.predict([tr_pairs[:,0], tr_pairs[:,1]])
train_accuracy = compute_accuracy(tr_y, y_pred_train)

y_pred_test = model.predict([ts_pairs[:,0], ts_pairs[:,1]])
test_accuracy = compute_accuracy(ts_y, y_pred_test)
```

```
print("Loss = {}, Train Accuracy = {} Test Accuracy = {}".format(loss,␣
↪train_accuracy, test_accuracy))
```

```
19980/19980 [==============================] - 1s 35us/sample - loss: 0.0621
Loss = 0.06214886149216734, Train Accuracy = 0.06121853642273712 Test Accuracy =
0.08498498498498498
```

[13]:
```
def plot_metrics(metric_name, title, ylim=5):
    plt.title(title)
    plt.ylim(0,ylim)
    plt.plot(history.history[metric_name],color='blue',label=metric_name)
    plt.plot(history.history['val_' + metric_name],color='green',label='val_' +␣
↪metric_name)


plot_metrics(metric_name='loss', title="Loss", ylim=0.2)
```

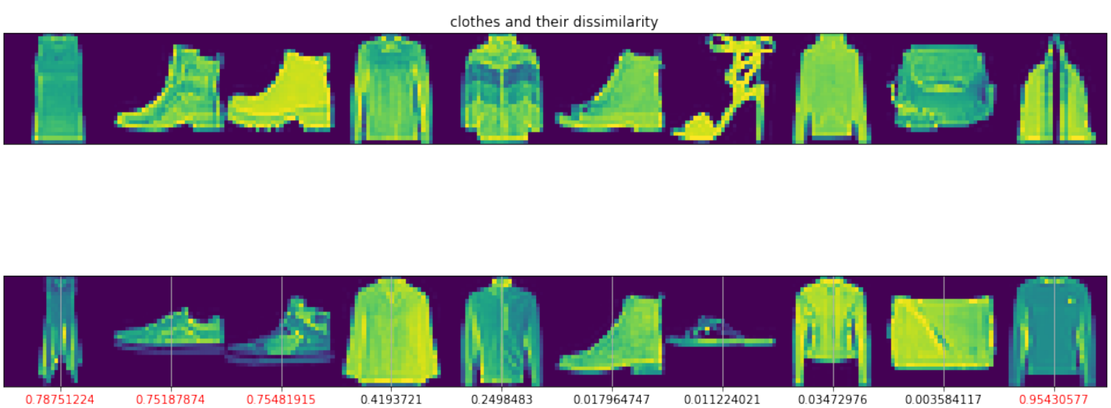


[14]:
```
# Matplotlib config
def visualize_images():
    plt.rc('image', cmap='gray_r')
    plt.rc('grid', linewidth=0)
    plt.rc('xtick', top=False, bottom=False, labelsize='large')
    plt.rc('ytick', left=False, right=False, labelsize='large')
    plt.rc('axes', facecolor='F8F8F8', titlesize="large", edgecolor='white')
```

```python
    plt.rc('text', color='a8151a')
    plt.rc('figure', facecolor='F0F0F0')# Matplotlib fonts


# utility to display a row of digits with their predictions
def display_images(left, right, predictions, labels, title, n):
    plt.figure(figsize=(17,3))
    plt.title(title)
    plt.yticks([])
    plt.xticks([])
    plt.grid(None)
    left = np.reshape(left, [n, 28, 28])
    left = np.swapaxes(left, 0, 1)
    left = np.reshape(left, [28, 28*n])
    plt.imshow(left)
    plt.figure(figsize=(17,3))
    plt.yticks([])
    plt.xticks([28*x+14 for x in range(n)], predictions)
    for i,t in enumerate(plt.gca().xaxis.get_ticklabels()):
        if predictions[i] > 0.5: t.set_color('red') # bad predictions in red
    plt.grid(None)
    right = np.reshape(right, [n, 28, 28])
    right = np.swapaxes(right, 0, 1)
    right = np.reshape(right, [28, 28*n])
    plt.imshow(right)
```

You can see sample results for 10 pairs of items below.

[15]:
```python
y_pred_train = np.squeeze(y_pred_train)
indexes = np.random.choice(len(y_pred_train), size=10)
display_images(tr_pairs[:, 0][indexes], tr_pairs[:, 1][indexes], y_pred_train[indexes], tr_y[indexes], "clothes and their dissimilarity", 10)
```



[ ]:

```
```