Siamese-Network-CIFAR10-Dataset

January 27, 2021

This example will go through creating and training a multi-input model. We will build a basic Siamese Network to find the similarity or dissimilarity between 32x32 images from the CIFAR10 dataset.

0.1 Imports

[1]:
```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Flatten, Dense, Dropout, Lambda,␣
↪Conv2D,MaxPooling2D
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.datasets import cifar10
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
```

0.2 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.

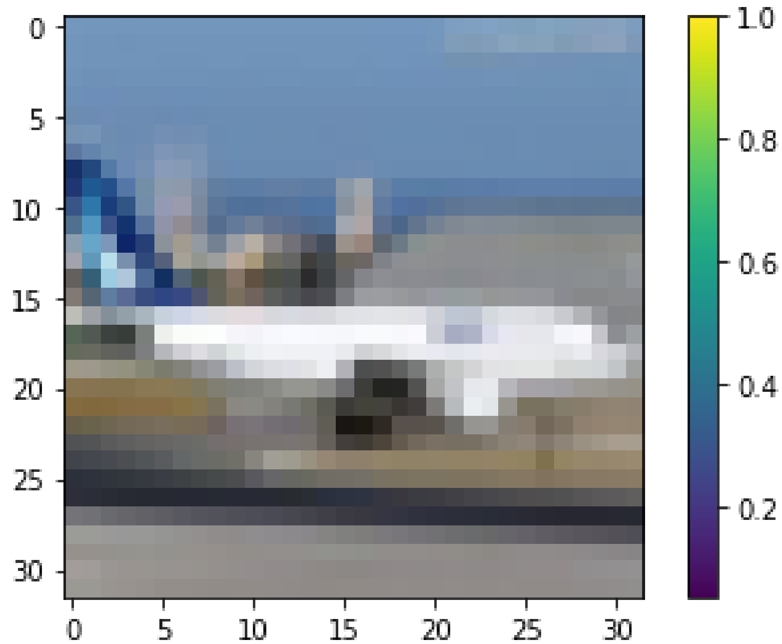
[3]:
```
# load the dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# 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)
```

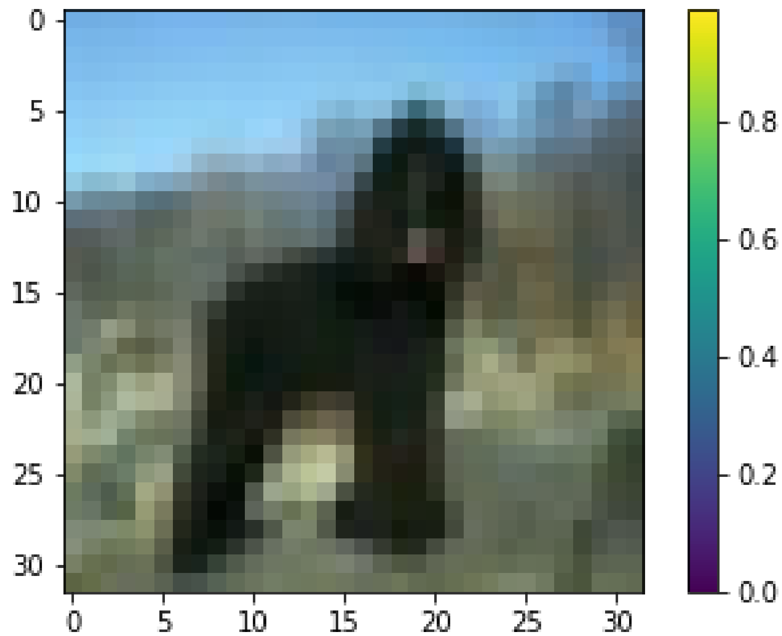
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 3s 0us/step
```

You can see a sample pair of images below.

[4]:

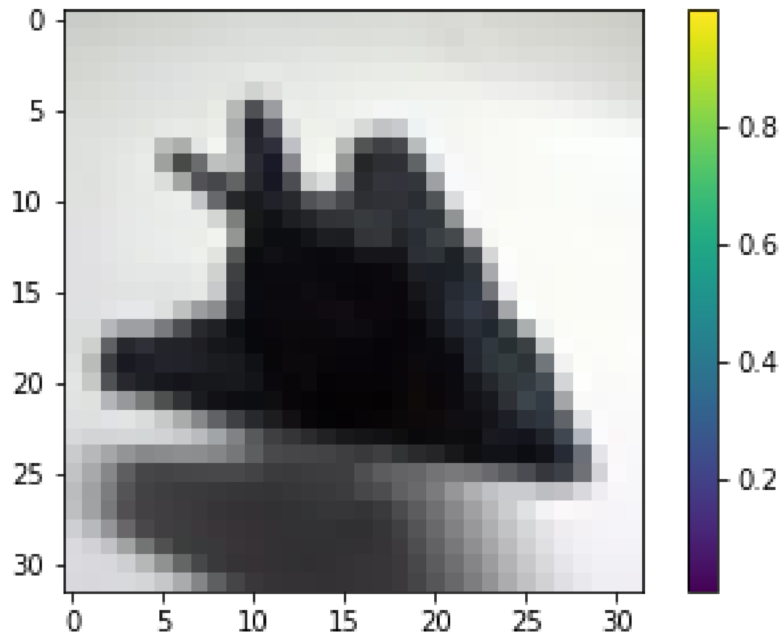
```
# array index
this_pair = 15

# 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(ts_y[this_pair])
```

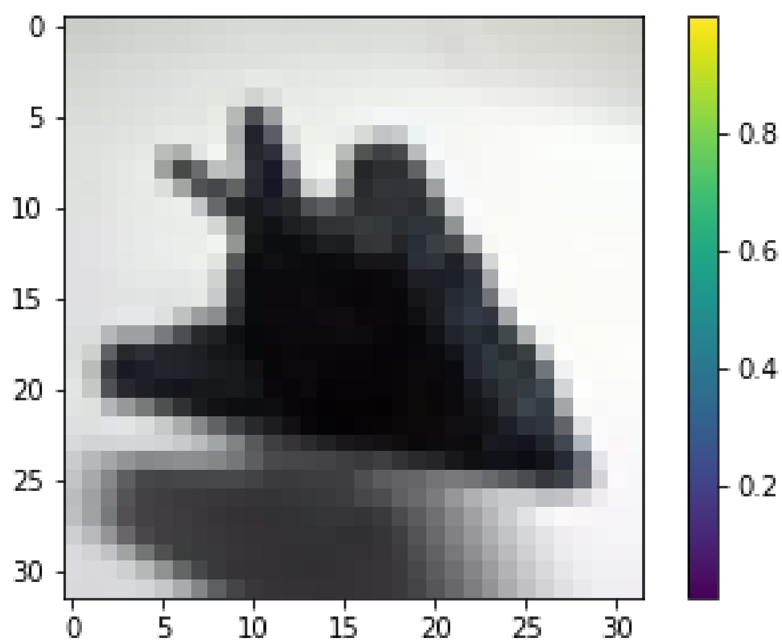


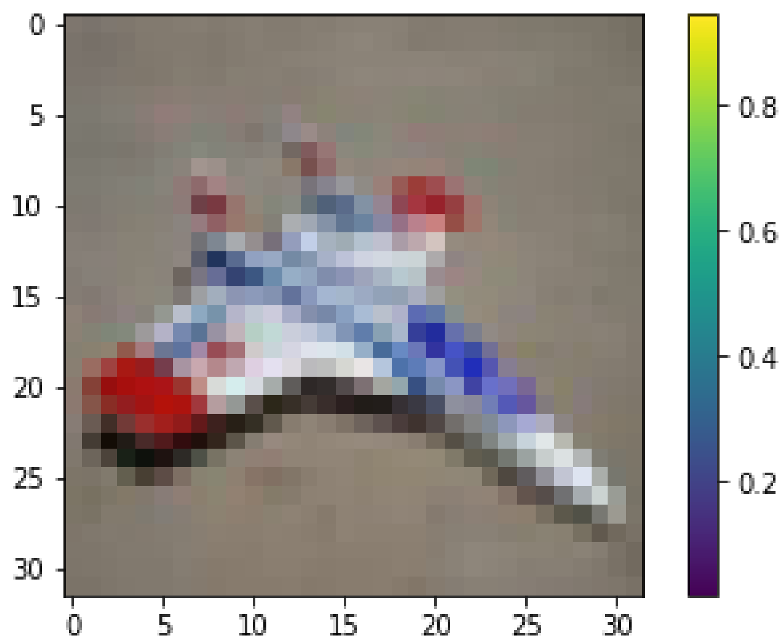


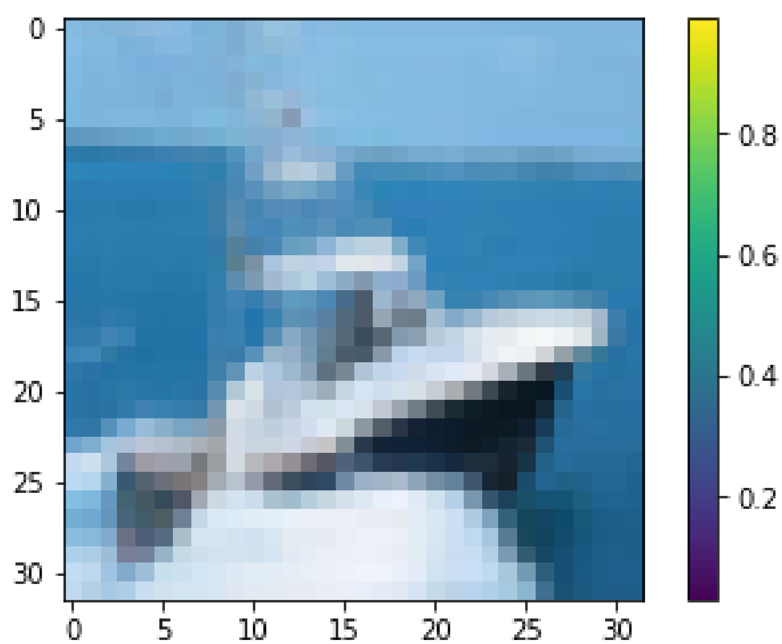

```
0.0
```

[6]:
```
# 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
30
0.8
0.6
0.4
0.2


0
5
10
15
20
25
30
0.8
0.6
0.4
0.2





0.3 Build the Model

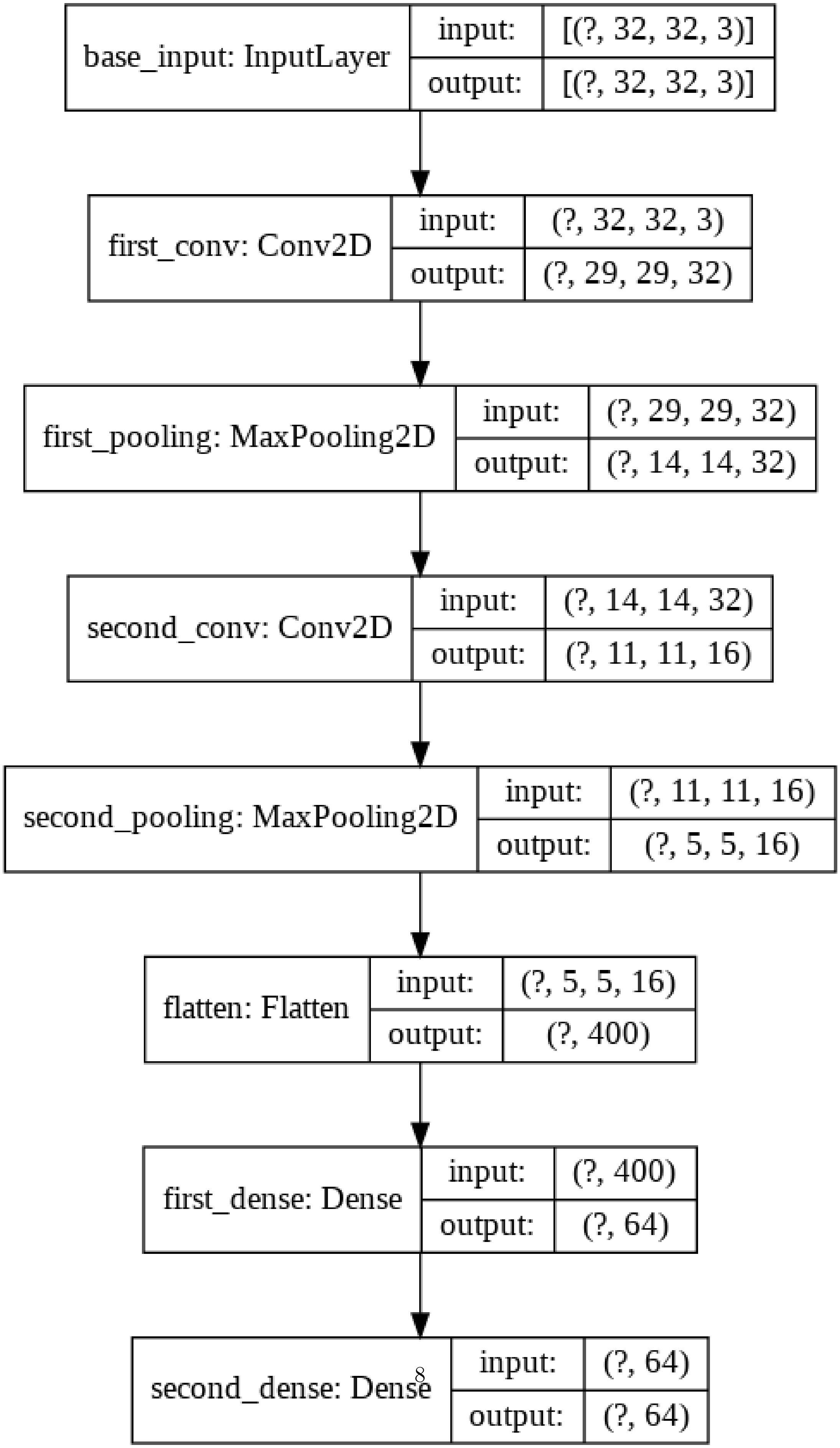
Next, we'll define some utilities for building our model.

[5]:
```python
def initialize_base_network():
    input = Input(shape=(32,32,3), name="base_input")
    #x = Flatten(name="flatten_input")(input)
    x = Conv2D(32, kernel_size=4, activation='relu',name="first_conv")(input)
    x = MaxPooling2D(pool_size=(2, 2),name="first_pooling")(x)
    x = Conv2D(16, kernel_size=4, activation='relu',name="second_conv")(x)
    x = MaxPooling2D(pool_size=(2, 2),name="second_pooling")(x)
    x = Flatten()(x)
    x = Dense(64, activation='relu',name="first_dense")(x)
    x = Dense(64, activation='relu',name="second_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.

[6]:
```python
base_network = initialize_base_network()
plot_model(base_network, show_shapes=True, show_layer_names=True,␣
↪to_file='base-model.png')
```

[6]:

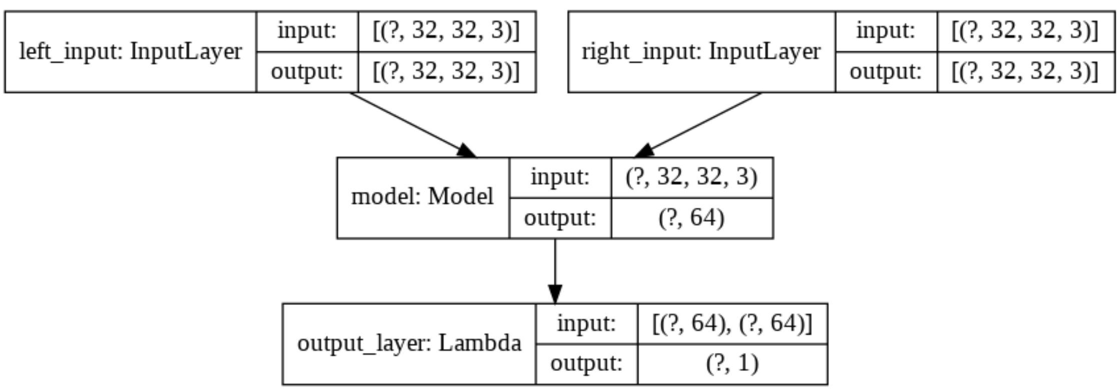


base_input: InputLayer
input: [(?, 32, 32, 3)]
output: [(?, 32, 32, 3)]
first_conv: Conv2D
input: (?, 32, 32, 3)
output: (?, 29, 29, 32)
first_pooling: MaxPooling2D
input: (?, 29, 29, 32)
output: (?, 14, 14, 32)
second_conv: Conv2D
input: (?, 14, 14, 32)
output: (?, 11, 11, 16)
second_pooling: MaxPooling2D
input: (?, 11, 11, 16)
output: (?, 5, 5, 16)
flatten: Flatten
input: (?, 5, 5, 16)
output: (?, 400)
first_dense: Dense
input: (?, 400)
output: (?, 64)
second_dense: Dense
input: (?, 64)
output: (?, 64)

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

[7]:
```
# create the left input and point to the base network
input_a = Input(shape=(32,32,3), name="left_input")
vect_output_a = base_network(input_a)

# create the right input and point to the base network
input_b = Input(shape=(32,32,3), 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')
```

[7]:



0.4 Train the Model

[8]:
```
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
```

[ ]:
```
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))
```

0.5 Model Evaluation

[ ]:
```
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)
```

[ ]:
```
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))
```

[ ]:
```
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=0.3)
```

[ ]:
```
# 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')
    plt.rc('text', color='a8151a')
```

```python
    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, 32, 32,-1])
    left = np.swapaxes(left, 0, 1)
    left = np.reshape(left, [32, 32*n,-1])
    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, 32, 32,-1])
    right = np.swapaxes(right, 0, 1)
    right = np.reshape(right, [32, 32*n,-1])
    plt.imshow(right)
```

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

```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], "Images and their dissimilarity", 10)
```