

In [1]:

```
1 # set tf 1.x for colab
2 %tensorflow_version 1.x
```

UsageError: Line magic function `%tensorflow_version` not found.

Denoising Autoencoders And Where To Find Them

Today we're going to train deep autoencoders and apply them to faces and similar images search.

Our new test subjects are human faces from the [lfw dataset \(http://vis-www.cs.umass.edu/lfw/\)](http://vis-www.cs.umass.edu/lfw/).

Import stuff

In [3]:

```
1 import sys
2 sys.path.append("../")
3 import grading
```

In [4]:

```
1 import tensorflow as tf
2 import keras, keras.layers as L, keras.backend as K
3 import numpy as np
4 from sklearn.model_selection import train_test_split
5 from lfw_dataset import load_lfw_dataset
6 %matplotlib inline
7 import matplotlib.pyplot as plt
8 import download_utils
9 import keras_utils
10 import numpy as np
11 from keras_utils import reset_tf_session
```

C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or 'ltype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
_np_qint8 = np.dtype(("qint8", np.int8, 1))
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or 'ltype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
_np_quint8 = np.dtype(("quint8", np.uint8, 1))
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or 'ltype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
_np_qint16 = np.dtype(("qint16", np.int16, 1))
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or 'ltype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
_np_quint16 = np.dtype(("quint16", np.uint16, 1))
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or 'ltype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
_np_qint32 = np.dtype(("qint32", np.int32, 1))
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or 'ltype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
_np_resource = np.dtype(("resource", np.ubyte, 1))
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or 'ltype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
_np_resource = np.dtype(("resource", np.ubyte, 1))

In [5]:

```
1 # !!! remember to clear session/graph if you rebuild your graph to avoid out-of-memory errors !!!
```

Load dataset

Dataset was downloaded for you. Relevant links (just in case):

- http://www.cs.columbia.edu/CAVE/databases/pubfig/download/lfw_attributes.txt (http://www.cs.columbia.edu/CAVE/databases/pubfig/download/lfw_attributes.txt)
- <http://vis-www.cs.umass.edu/lfw/lfw-deepfunneled.tgz> (<http://vis-www.cs.umass.edu/lfw/lfw-deepfunneled.tgz>)
- <http://vis-www.cs.umass.edu/lfw/lfw.tgz> (<http://vis-www.cs.umass.edu/lfw/lfw.tgz>)

In [6]:

```
1 # we downloaded them for you, just link them here
2 download_utils.link_week_4_resources()
```

In [7]:

```
1 # load images
2 X, attr = load_lfw_dataset(use_raw=True, dimx=32, dimy=32)
3 IMG_SHAPE = X.shape[1:]
4
5 # center images
6 X = X.astype('float32') / 255.0 - 0.5
7
8 # split
9 X_train, X_test = train_test_split(X, test_size=0.1, random_state=42)
```

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

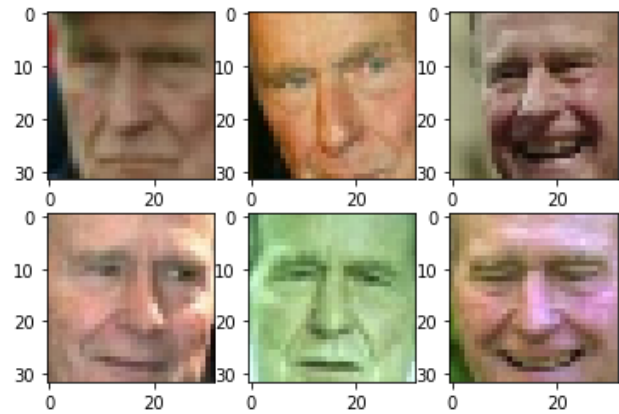
In [8]:

```
1 def show_image(x):
2     plt.imshow(np.clip(x + 0.5, 0, 1))
```

```
In [9]: 1 plt.title('sample images')
2
3 for i in range(6):
4     plt.subplot(2,3,i+1)
5     show_image(X[i])
6
7 print("X shape:", X.shape)
8 print("attr shape:", attr.shape)
9
10 # try to free memory
11 del X
12 import gc
13 gc.collect()
```

X shape: (13143, 32, 32, 3)
attr shape: (13143, 73)

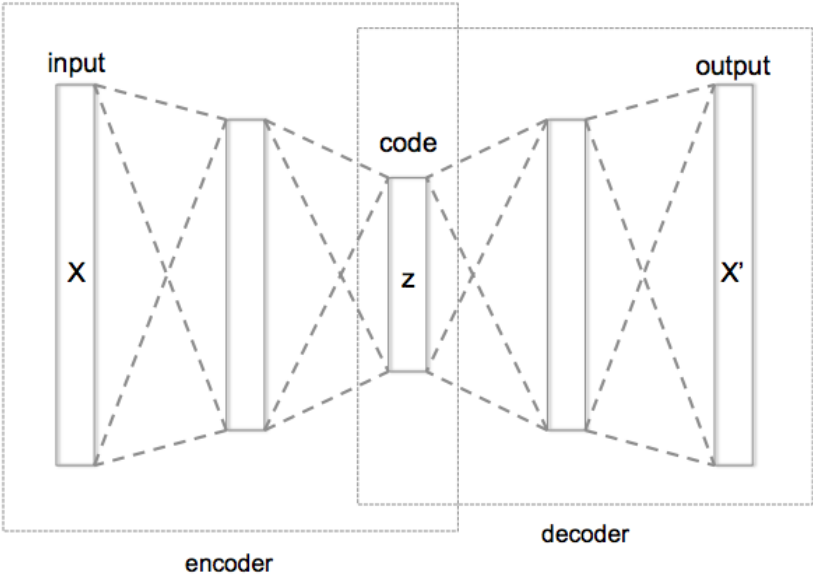
Out[9]: 0



Autoencoder architecture

Let's design autoencoder as two sequential keras models: the encoder and decoder respectively.

We will then use symbolic API to apply and train these models.



First step: PCA

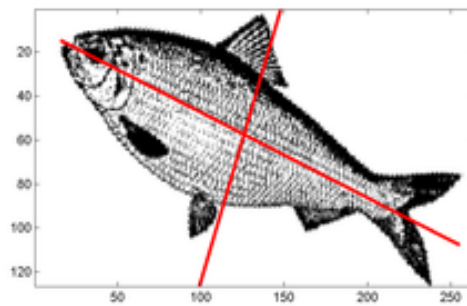
Principal Component Analysis is a popular dimensionality reduction method.

Under the hood, PCA attempts to decompose object-feature matrix X into two smaller matrices: W and \hat{W} minimizing _mean squared error_:

$$\| (XW)\hat{W} - X \|_2^2 \rightarrow_{W, \hat{W}} \min$$

- $X \in \mathbb{R}^{n \times m}$ - object matrix (**centered**);
- $W \in \mathbb{R}^{m \times d}$ - matrix of direct transformation;
- $\hat{W} \in \mathbb{R}^{d \times m}$ - matrix of reverse transformation;
- n samples, m original dimensions and d target dimensions;

In geometric terms, we want to find d axes along which most of variance occurs. The "natural" axes, if you wish.



PCA can also be seen as a special case of an autoencoder.

- **Encoder:** $X \rightarrow \text{Dense}(d \text{ units}) \rightarrow \text{code}$
- **Decoder:** $\text{code} \rightarrow \text{Dense}(m \text{ units}) \rightarrow X$

Where Dense is a fully-connected layer with linear activation: $f(X) = W \cdot X + \vec{b}$

Note: the bias term in those layers is responsible for "centering" the matrix i.e. subtracting mean.

```
In [10]: 1 def build_pca_autoencoder(img_shape, code_size):
2         """
3         Here we define a simple linear autoencoder as described above.
4         We also flatten and un-flatten data to be compatible with image shapes
5         """
6
7         encoder = keras.models.Sequential()
8         encoder.add(L.InputLayer(img_shape))
9         encoder.add(L.Flatten())           #flatten image to vector
10        encoder.add(L.Dense(code_size))    #actual encoder
11
12        decoder = keras.models.Sequential()
13        decoder.add(L.InputLayer((code_size,)))
14        decoder.add(L.Dense(np.prod(img_shape))) #actual decoder, height*width*3 units
15        decoder.add(L.Reshape(img_shape))    #un-flatten
16
17        return encoder,decoder
```

Meld them together into one model:

```
In [11]: 1 s = reset_tf_session()
2
3 encoder, decoder = build_pca_autoencoder(IMG_SHAPE, code_size=32)
4
5 inp = L.Input(IMG_SHAPE)
6 code = encoder(inp)
7 reconstruction = decoder(code)
8
9 autoencoder = keras.models.Model(inputs=inp, outputs=reconstruction)
10 autoencoder.compile(optimizer='adamax', loss='mse')
11
12 autoencoder.fit(x=X_train, y=X_train, epochs=15,
13               validation_data=[X_test, X_test],
14               callbacks=[keras_utils.TqdmProgressCallback()],
15               verbose=0)
```

WARNING:tensorflow:From ..\keras_utils.py:68: The name tf.get_default_session is deprecated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:95: The name tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:98: The name tf.placeholder_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:102: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From ..\keras_utils.py:75: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto instead.

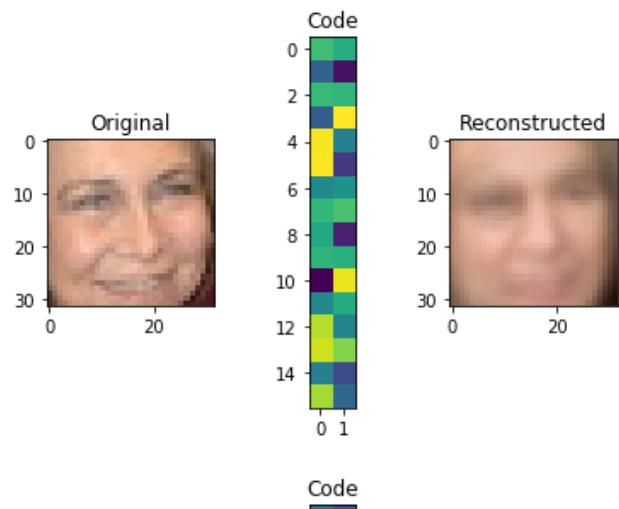
WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\optimizers.py:790: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/15

```
In [12]: 1 def visualize(img,encoder,decoder):
2         """Draws original, encoded and decoded images"""
3         code = encoder.predict(img[None])[0] # img[None] is the same as img[np.newaxis, :]
4         reco = decoder.predict(code[None])[0]
5
6         plt.subplot(1,3,1)
7         plt.title("Original")
8         show_image(img)
9
10        plt.subplot(1,3,2)
11        plt.title("Code")
12        plt.imshow(code.reshape([code.shape[-1]//2,-1]))
13
14        plt.subplot(1,3,3)
15        plt.title("Reconstructed")
16        show_image(reco)
17        plt.show()
18
```

```
In [13]: 1 score = autoencoder.evaluate(X_test,X_test,verbose=0)
2 print("PCA MSE:", score)
3
4 for i in range(5):
5     img = X_test[i]
6     visualize(img,encoder,decoder)
```

PCA MSE: 0.006625801430016529



Going deeper: convolutional autoencoder

PCA is neat but surely we can do better. This time we want you to build a deep convolutional autoencoder by... stacking more layers.

Encoder

The **encoder** part is pretty standard, we stack convolutional and pooling layers and finish with a dense layer to get the representation of desirable size (`code_size`).

We recommend to use `activation='elu'` for all convolutional and dense layers.

We recommend to repeat (conv, pool) 4 times with kernel size (3, 3), `padding='same'` and the following numbers of output channels: 32, 64, 128, 256 .

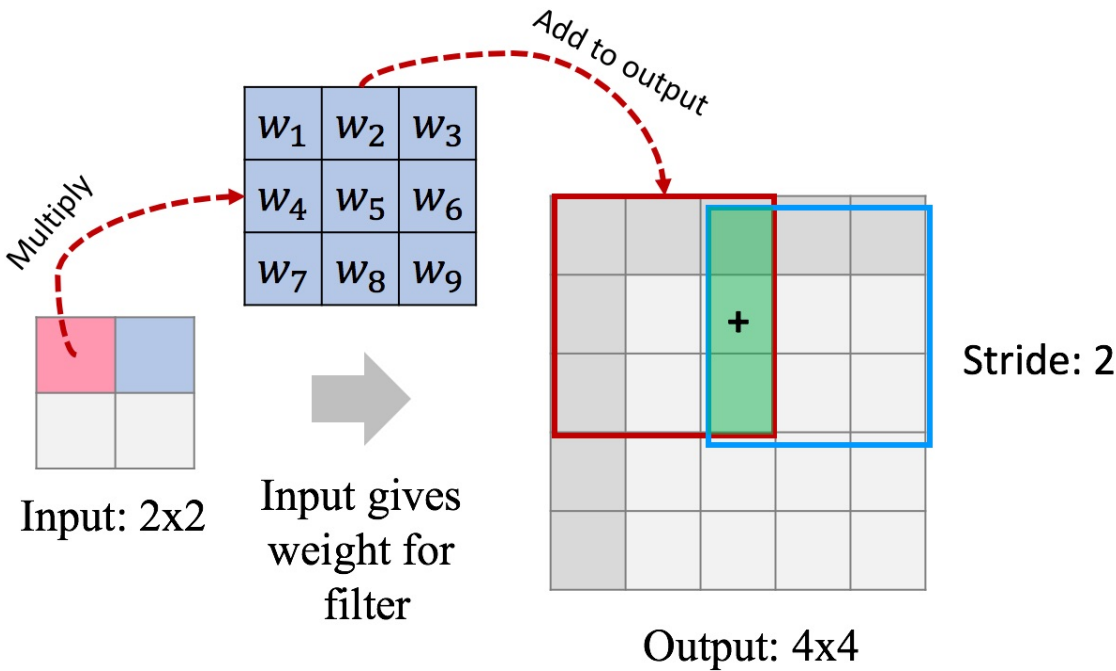
Remember to flatten (`L.Flatten()`) output before adding the last dense layer!

Decoder

For **decoder** we will use so-called "transpose convolution".

Traditional convolutional layer takes a patch of an image and produces a number (patch -> number). In "transpose convolution" we want to take a number and produce a patch of an image (number -> patch). We need this layer to "undo" convolutions in encoder. We had a glimpse of it during week 3 (watch [this video \(https://www.coursera.org/learn/intro-to-deep-learning/lecture/auRqf/a-glimpse-of-other-computer-vision-tasks\)](https://www.coursera.org/learn/intro-to-deep-learning/lecture/auRqf/a-glimpse-of-other-computer-vision-tasks) starting at 5:41).

Here's how "transpose convolution" works:



In this example we use a stride of 2 to produce 4x4 output, this way we "undo" pooling as well. Another way to think about it: we "undo" convolution with stride 2 (which is similar to conv + pool).

You can add "transpose convolution" layer in Keras like this:

```
L.Conv2DTranspose(filters=?, kernel_size=(3, 3), strides=2, activation='elu', padding='same')
```

Our decoder starts with a dense layer to "undo" the last layer of encoder. Remember to reshape its output to "undo" `L.Flatten()` in encoder.

Now we're ready to undo (conv, pool) pairs. For this we need to stack 4 `L.Conv2DTranspose` layers with the following numbers of output channels: 128, 64, 32, 3 . Each of these layers will learn to "undo" (conv, pool) pair in encoder. For the last `L.Conv2DTranspose` layer use `activation=None` because that is our final image.

```

In [14]: 1 # Let's play around with transpose convolution on examples first
2 def test_conv2d_transpose(img_size, filter_size):
3     print("Transpose convolution test for img_size={}, filter_size={}".format(img_size, filter_size))
4
5     x = (np.arange(img_size ** 2, dtype=np.float32) + 1).reshape((1, img_size, img_size, 1))
6     f = (np.ones(filter_size ** 2, dtype=np.float32)).reshape((filter_size, filter_size, 1, 1))
7
8     s = reset_tf_session()
9
10    conv = tf.nn.conv2d_transpose(x, f,
11                                output_shape=(1, img_size * 2, img_size * 2, 1),
12                                strides=[1, 2, 2, 1],
13                                padding='SAME')
14
15    result = s.run(conv)
16    print("input:")
17    print(x[0, :, :, 0])
18    print("filter:")
19    print(f[:, :, 0, 0])
20    print("output:")
21    print(result[0, :, :, 0])
22    s.close()
23
24 test_conv2d_transpose(img_size=2, filter_size=2)
25 test_conv2d_transpose(img_size=2, filter_size=3)
26 test_conv2d_transpose(img_size=4, filter_size=2)
27 test_conv2d_transpose(img_size=4, filter_size=3)

```

Transpose convolution test for img_size=2, filter_size=2:

input:

```
[[1. 2.]
 [3. 4.]]
```

filter:

```
[[1. 1.]
 [1. 1.]]
```

output:

```
[[1. 1. 2. 2.]
 [1. 1. 2. 2.]
 [3. 3. 4. 4.]
 [3. 3. 4. 4.]]
```

Transpose convolution test for img_size=2, filter_size=3:

input:

```
[[1. 2.]
 [3. 4.]]
```

filter:

```
[[1. 1. 1.]
 [1. 1. 1.]
 -. -. -.]
```

```

In [15]: 1 def build_deep_autoencoder(img_shape, code_size):
2     """PCA's deeper brother. See instructions above. Use `code_size` in layer definitions."""
3     H,W,C = img_shape
4
5     # encoder
6     encoder = keras.models.Sequential()
7     encoder.add(L.InputLayer(img_shape))
8
9     ### YOUR CODE HERE: define encoder as per instructions above ###
10    encoder.add(L.Conv2D(filters=32, kernel_size=(3,3), padding='same', activation='elu'))
11    encoder.add(L.MaxPooling2D(pool_size=(2,2)))
12    encoder.add(L.Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='elu'))
13    encoder.add(L.MaxPooling2D(pool_size=(2,2)))
14    encoder.add(L.Conv2D(filters=128, kernel_size=(3,3), padding='same', activation='elu'))
15    encoder.add(L.MaxPooling2D(pool_size=(2,2)))
16    encoder.add(L.Conv2D(filters=256, kernel_size=(3,3), padding='same', activation='elu'))
17    encoder.add(L.MaxPooling2D(pool_size=(2,2)))
18    encoder.add(L.Flatten())
19    encoder.add(L.Dense(code_size, activation='elu'))
20
21    # decoder
22    decoder = keras.models.Sequential()
23    decoder.add(L.InputLayer((code_size,)))
24
25    ### YOUR CODE HERE: define decoder as per instructions above ###
26    conv_shape=np.floor_divide(img_shape[:2], 2**4)
27    decoder.add(L.Dense(np.prod(conv_shape)*256, activation='elu'))
28    target_shape = tuple(conv_shape) + (256, )
29    decoder.add(L.Reshape(target_shape))
30    decoder.add(L.Conv2DTranspose(filters=128, kernel_size=(3,3), strides=2, activation='elu', padding='same'))
31    decoder.add(L.Conv2DTranspose(filters=64, kernel_size=(3,3), strides=2, activation='elu', padding='same'))
32    decoder.add(L.Conv2DTranspose(filters=32, kernel_size=(3,3), strides=2, activation='elu', padding='same'))
33    decoder.add(L.Conv2DTranspose(filters=3, kernel_size=(3,3), strides=2, activation=None, padding='same'))
34
35    return encoder, decoder

```

```
In [16]: 1 # Check autoencoder shapes along different code_sizes
2 get_dim = lambda layer: np.prod(layer.output_shape[1:])
3 for code_size in [1,8,32,128,512]:
4     s = reset_tf_session()
5     encoder, decoder = build_deep_autoencoder(IMG_SHAPE, code_size=code_size)
6     print("Testing code size %i" % code_size)
7     assert encoder.output_shape[1:]==(code_size,), "encoder must output a code of required size"
8     assert decoder.output_shape[1:]==IMG_SHAPE, "decoder must output an image of valid shape"
9     assert len(encoder.trainable_weights)>=6, "encoder must contain at least 3 layers"
10    assert len(decoder.trainable_weights)>=6, "decoder must contain at least 3 layers"
11
12    for layer in encoder.layers + decoder.layers:
13        assert get_dim(layer) >= code_size, "Encoder layer %s is smaller than bottleneck (%i units)"%(layer.name,get_dim(layer))
14
15    print("All tests passed!")
16    s = reset_tf_session()
```

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:3976: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

Testing code size 1
Testing code size 8
Testing code size 32
Testing code size 128
Testing code size 512
All tests passed!

```
In [17]: 1 # Look at encoder and decoder shapes.
2 # Total number of trainable parameters of encoder and decoder should be close.
3 s = reset_tf_session()
4 encoder, decoder = build_deep_autoencoder(IMG_SHAPE, code_size=32)
5 encoder.summary()
6 decoder.summary()
```

Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896

max_pooling2d_1 (MaxPooling2)	(None, 16, 16, 32)	0

conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496

max_pooling2d_2 (MaxPooling2)	(None, 8, 8, 64)	0

conv2d_3 (Conv2D)	(None, 8, 8, 128)	73856

max_pooling2d_3 (MaxPooling2)	(None, 4, 4, 128)	0

conv2d_4 (Conv2D)	(None, 4, 4, 256)	295168

max_pooling2d_4 (MaxPooling2)	(None, 2, 2, 256)	0

Convolutional autoencoder training. This will take **1 hour**. You're aiming at ~0.0056 validation MSE and ~0.0054 training MSE.

```
In [18]: 1 s = reset_tf_session()
2
3 encoder, decoder = build_deep_autoencoder(IMG_SHAPE, code_size=32)
4
5 inp = L.Input(IMG_SHAPE)
6 code = encoder(inp)
7 reconstruction = decoder(code)
8
9 autoencoder = keras.models.Model(inputs=inp, outputs=reconstruction)
10 autoencoder.compile(optimizer="adamax", loss='mse')
```

```
In [19]: 1 # we will save model checkpoints here to continue training in case of kernel death
2 model_filename = 'autoencoder.{0:03d}.hdf5'
3 last_finished_epoch = None
4
5 ##### uncomment below to continue training from model checkpoint
6 ##### fill `last_finished_epoch` with your latest finished epoch
7 # from keras.models import load_model
8 # s = reset_tf_session()
9 # last_finished_epoch = 4
10 # autoencoder = load_model(model_filename.format(last_finished_epoch))
11 # encoder = autoencoder.layers[1]
12 # decoder = autoencoder.layers[2]
```



```
In [20]: 1 autoencoder.fit(x=X_train, y=X_train, epochs=25,
2               validation_data=[X_test, X_test],
3               callbacks=[keras_utils.ModelSaveCallback(model_filename),
4                       keras_utils.TqdmProgressCallback()],
5               verbose=0,
6               initial_epoch=last_finished_epoch or 0)
```

Epoch 1/25

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Model saved in autoencoder.000.hdf5

Epoch 2/25

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

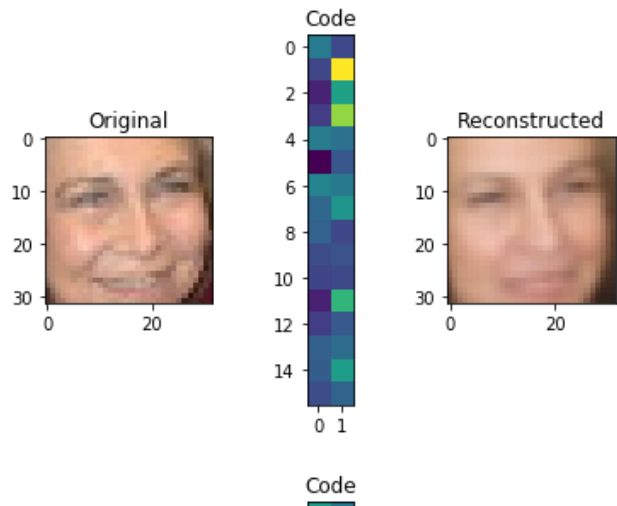
Model saved in autoencoder.001.hdf5

Epoch 3/25

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget

```
In [21]: 1 reconstruction_mse = autoencoder.evaluate(X_test, X_test, verbose=0)
2 print("Convolutional autoencoder MSE:", reconstruction_mse)
3 for i in range(5):
4     img = X_test[i]
5     visualize(img, encoder, decoder)
```

Convolutional autoencoder MSE: 0.005612965745447706



```
In [22]: 1 # save trained weights
2 encoder.save_weights("encoder.h5")
3 decoder.save_weights("decoder.h5")
```

```
In [23]: 1 # restore trained weights
2 s = reset_tf_session()
3
4 encoder, decoder = build_deep_autoencoder(IMG_SHAPE, code_size=32)
5 encoder.load_weights("encoder.h5")
6 decoder.load_weights("decoder.h5")
7
8 inp = L.Input(IMG_SHAPE)
9 code = encoder(inp)
10 reconstruction = decoder(code)
11
12 autoencoder = keras.models.Model(inputs=inp, outputs=reconstruction)
13 autoencoder.compile(optimizer="adamax", loss='mse')
14
15 print(autoencoder.evaluate(X_test, X_test, verbose=0))
16 print(reconstruction_mse)
```

0.005612965722784343
0.005612965745447706

Submit to Coursera

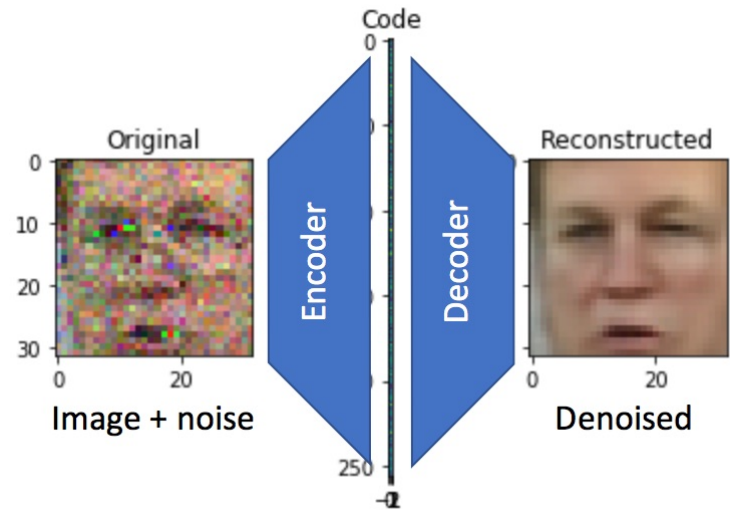
```
In [24]: 1 from submit import submit_autoencoder
2 submission = build_deep_autoencoder(IMG_SHAPE, code_size=71)
3
4 # token expires every 30 min
5 COURSERA_TOKEN = "YCLb6IbDdpLrFnIf"
6 COURSERA_EMAIL = "lxwvictor@gmail.com"
7
8 submit_autoencoder(submission, reconstruction_mse, COURSERA_EMAIL, COURSERA_TOKEN)
```

Submitted to Coursera platform. See results on assignment page!

Optional: Denoising Autoencoder

This part is **optional**, it shows you one useful application of autoencoders: denoising. You can run this code and make sure denoising works :)

Let's now turn our model into a denoising autoencoder:



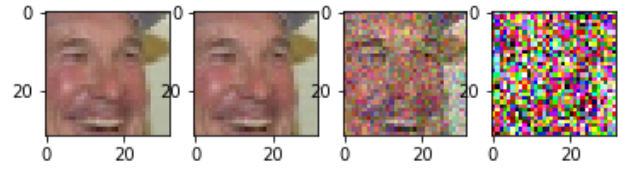
We'll keep the model architecture, but change the way it is trained. In particular, we'll corrupt its input data randomly with noise before each epoch.

There are many strategies to introduce noise: adding gaussian white noise, occluding with random black rectangles, etc. We will add gaussian white noise.

```
In [25]: 1 def apply_gaussian_noise(X,sigma=0.1):
2         """
3         adds noise from standard normal distribution with standard deviation sigma
4         :param X: image tensor of shape [batch,height,width,3]
5         Returns X + noise.
6         """
7         ### YOUR CODE HERE ###
8         size, row, col, ch = X.shape
9         mean = 0
10        gauss = np.random.normal(mean, sigma, (size, row, col, ch))
11        noise = gauss.reshape(size, row, col, ch)
12        return X + noise
```

```
In [26]: 1 # noise tests
2 theoretical_std = (X_train[:100].std()**2 + 0.5**2)**.5
3 our_std = apply_gaussian_noise(X_train[:100],sigma=0.5).std()
4 assert abs(theoretical_std - our_std) < 0.01, "Standard deviation does not match it's required value. Make sure you use sigma as :
5 assert abs(apply_gaussian_noise(X_train[:100],sigma=0.5).mean() - X_train[:100].mean()) < 0.01, "Mean has changed. Please add zero"
```

```
In [27]: 1 # test different noise scales
2 plt.subplot(1,4,1)
3 show_image(X_train[0])
4 plt.subplot(1,4,2)
5 show_image(apply_gaussian_noise(X_train[:1],sigma=0.01)[0])
6 plt.subplot(1,4,3)
7 show_image(apply_gaussian_noise(X_train[:1],sigma=0.1)[0])
8 plt.subplot(1,4,4)
9 show_image(apply_gaussian_noise(X_train[:1],sigma=0.5)[0])
```



Training will take **1 hour**.


```
In [28]: 1 s = reset_tf_session()
2
3 # we use bigger code size here for better quality
4 encoder, decoder = build_deep_autoencoder(IMG_SHAPE, code_size=512)
5 assert encoder.output_shape[1:]==(512,), "encoder must output a code of required size"
6
7 inp = L.Input(IMG_SHAPE)
8 code = encoder(inp)
9 reconstruction = decoder(code)
10
11 autoencoder = keras.models.Model(inp, reconstruction)
12 autoencoder.compile('adamax', 'mse')
13
14 for i in range(25):
15     print("Epoch %i/25, Generating corrupted samples..."%(i+1))
16     X_train_noise = apply_gaussian_noise(X_train)
17     X_test_noise = apply_gaussian_noise(X_test)
18
19     # we continue to train our model with new noise-augmented data
20     autoencoder.fit(x=X_train_noise, y=X_train, epochs=1,
21                   validation_data=(X_test_noise, X_test),
22                   callbacks=[keras_utils.TqdmProgressCallback()],
23                   verbose=0)
```

Epoch 1/25, Generating corrupted samples...

Epoch 1/1

A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Epoch 2/25, Generating corrupted samples...

Epoch 1/1

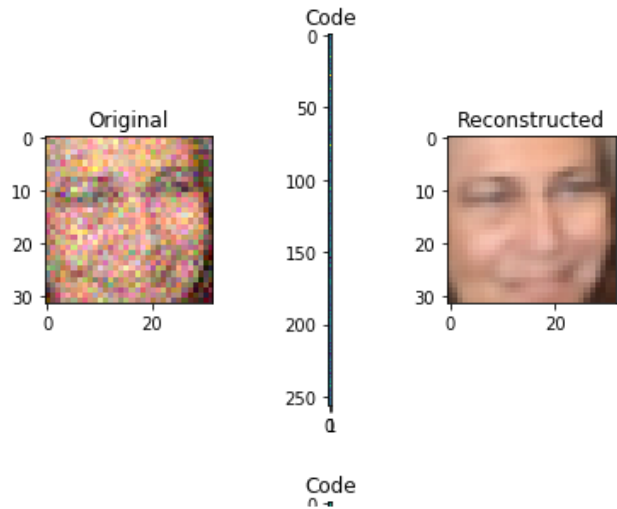
A Jupyter widget could not be displayed because the widget state could not be found. This could happen if the kernel storing the widget is no longer available, or if the widget state was not saved in the notebook. You may be able to create the widget by running the appropriate cells.

Epoch 3/25, Generating corrupted samples...

Epoch 1/1

```
In [29]: 1 X_test_noise = apply_gaussian_noise(X_test)
2 denoising_mse = autoencoder.evaluate(X_test_noise, X_test, verbose=0)
3 print("Denoising MSE:", denoising_mse)
4 for i in range(5):
5     img = X_test_noise[i]
6     visualize(img,encoder,decoder)
```

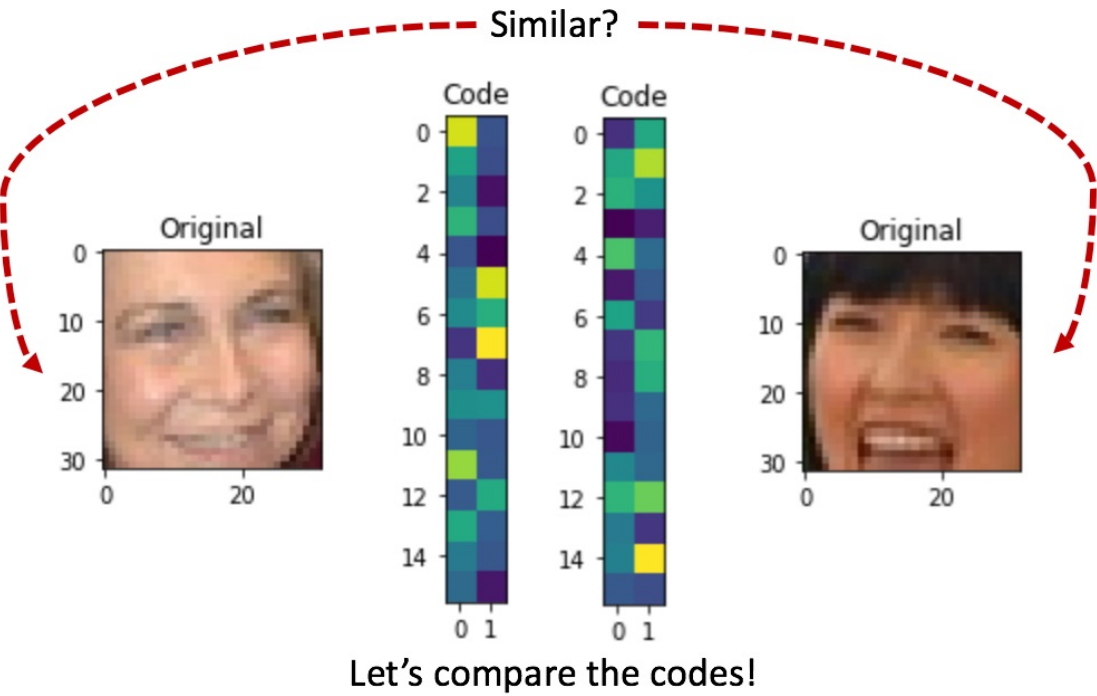
Denoising MSE: 0.0028226802647007567



Optional: Image retrieval with autoencoders

So we've just trained a network that converts image into itself imperfectly. This task is not that useful in and of itself, but it has a number of awesome side-effects. Let's see them in action.

First thing we can do is image retrieval aka image search. We will give it an image and find similar images in latent space:



To speed up retrieval process, one should use Locality Sensitive Hashing on top of encoded vectors. This [technique \(https://erikbern.com/2015/07/04/benchmark-of-approximate-nearest-neighbor-libraries.html\)](https://erikbern.com/2015/07/04/benchmark-of-approximate-nearest-neighbor-libraries.html) can narrow down the potential nearest neighbours of our image in latent space (encoder code). We will caculate nearest neighbours in brute force way for simplicity.

```
In [30]: 1 # restore trained encoder weights
2 s = reset_tf_session()
3 encoder, decoder = build_deep_autoencoder(IMG_SHAPE, code_size=32)
4 encoder.load_weights("encoder.h5")
```

```
In [31]: 1 images = X_train
2 ### YOUR CODE HERE: encode all images ###
3 codes = encoder.predict(images)
4 assert len(codes) == len(images)
```

```
In [32]: 1 from sklearn.neighbors.unsupervised import NearestNeighbors
2 nei_clf = NearestNeighbors(metric="euclidean")
3 nei_clf.fit(codes)
```

C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning: The sklearn.neighbors.unsupervised module is deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.neighbors. Anything that cannot be imported from sklearn.neighbors is now part of the private API.
warnings.warn(message, FutureWarning)

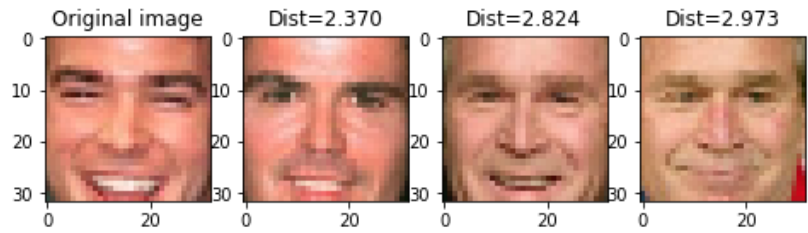
```
Out[32]: NearestNeighbors(algorithm='auto', leaf_size=30, metric='euclidean',
metric_params=None, n_jobs=None, n_neighbors=5, p=2,
radius=1.0)
```

```
In [33]: 1 def get_similar(image, n_neighbors=5):
2     assert image.ndim==3,"image must be [batch,height,width,3]"
3
4     code = encoder.predict(image[None])
5
6     (distances,),(idx,) = nei_clf.kneighbors(code,n_neighbors=n_neighbors)
7
8     return distances,images[idx]
```

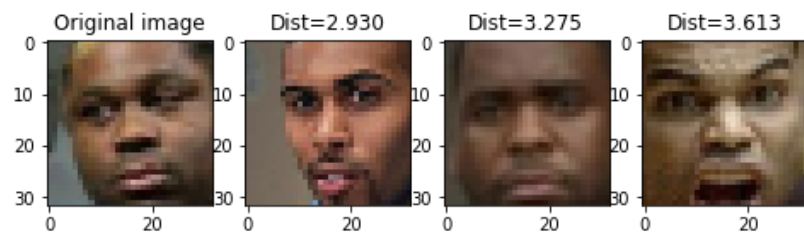
```
In [34]: 1 def show_similar(image):
2
3     distances,neighbors = get_similar(image,n_neighbors=3)
4
5     plt.figure(figsize=[8,7])
6     plt.subplot(1,4,1)
7     show_image(image)
8     plt.title("Original image")
9
10    for i in range(3):
11        plt.subplot(1,4,i+2)
12        show_image(neighbors[i])
13        plt.title("Dist=%.3f"%distances[i])
14    plt.show()
```

Cherry-picked examples:

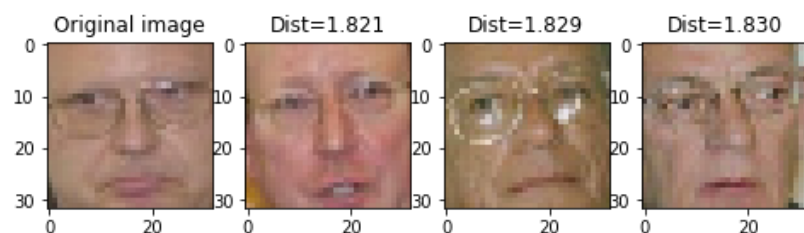
```
In [35]: 1 # smiles
2 show_similar(X_test[247])
```



```
In [36]: 1 # ethnicity
2 show_similar(X_test[56])
```



```
In [37]: 1 # glasses
2 show_similar(X_test[63])
```

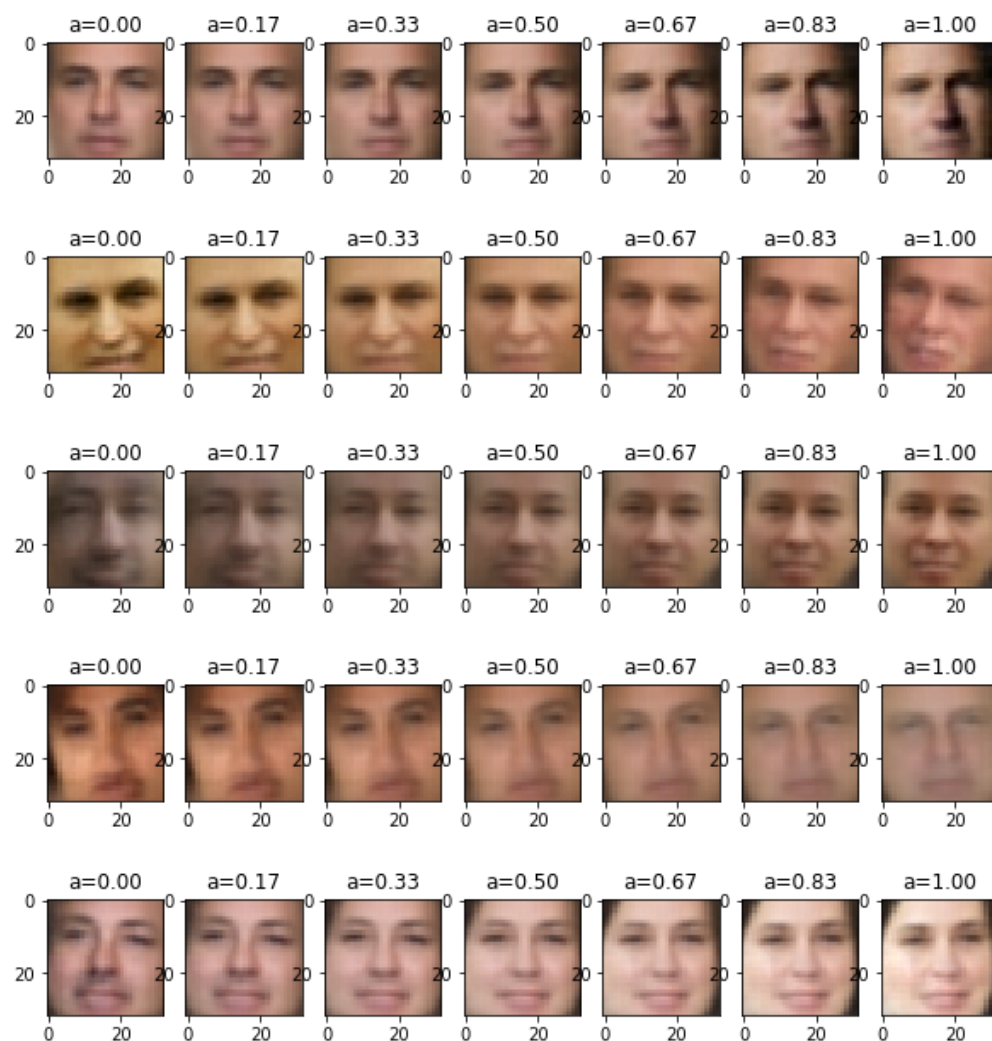


Optional: Cheap image morphing

We can take linear combinations of image codes to produce new images with decoder.

```
In [38]: 1 # restore trained encoder weights
2 s = reset_tf_session()
3 encoder, decoder = build_deep_autoencoder(IMG_SHAPE, code_size=32)
4 encoder.load_weights("encoder.h5")
5 decoder.load_weights("decoder.h5")
```

```
In [39]: 1 for _ in range(5):
2     image1, image2 = X_test[np.random.randint(0, len(X_test), size=2)]
3
4     code1, code2 = encoder.predict(np.stack([image1, image2]))
5
6     plt.figure(figsize=[10,4])
7     for i,a in enumerate(np.linspace(0,1,num=7)):
8
9         output_code = code1*(1-a) + code2*(a)
10        output_image = decoder.predict(output_code[None])[0]
11
12        plt.subplot(1,7,i+1)
13        show_image(output_image)
14        plt.title("a=%.2f"%a)
15
16    plt.show()
```



That's it!

Of course there's a lot more you can do with autoencoders.

If you want to generate images from scratch, however, we recommend you our honor track on Generative Adversarial Networks or GANs.

