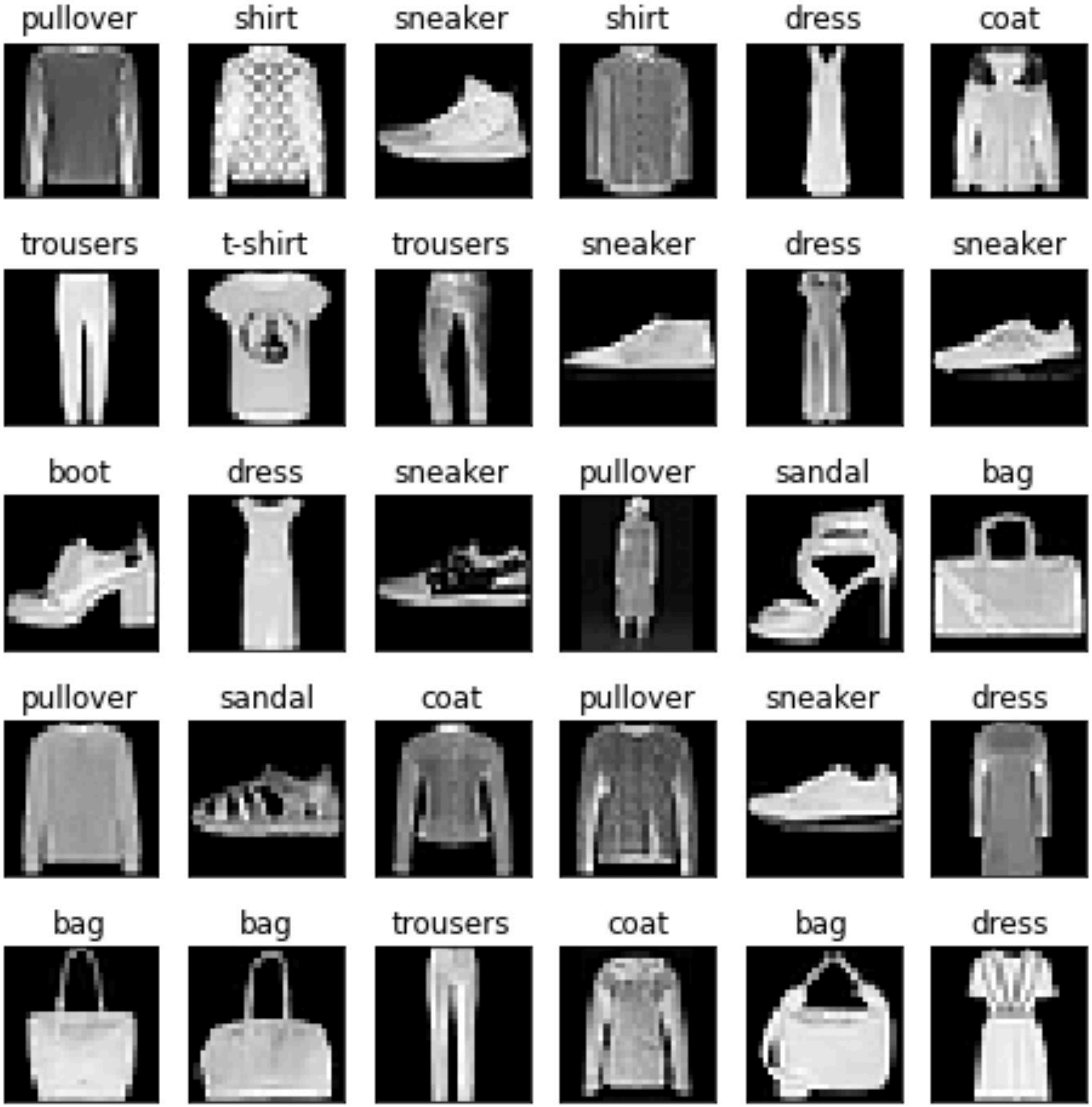
Homework 7 (due Friday Nov 11)

introduce Q1 today (will introduce Q2 on Thu)

This assignment asks you to create multi-layer networks that learn to classify images of clothing. Fashion-MNIST is a dataset of clothing images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

see `Homework7_2022.ipynb`


pullover
shirt
sneaker
shirt
dress
coat
trousers
t-shirt
trousers
sneaker
dress
sneaker
boot
dress
sneaker
pullover
sandal
bag
pullover
sandal
coat
pullover
sneaker
dress
bag
bag
trousers
coat
bag
dress

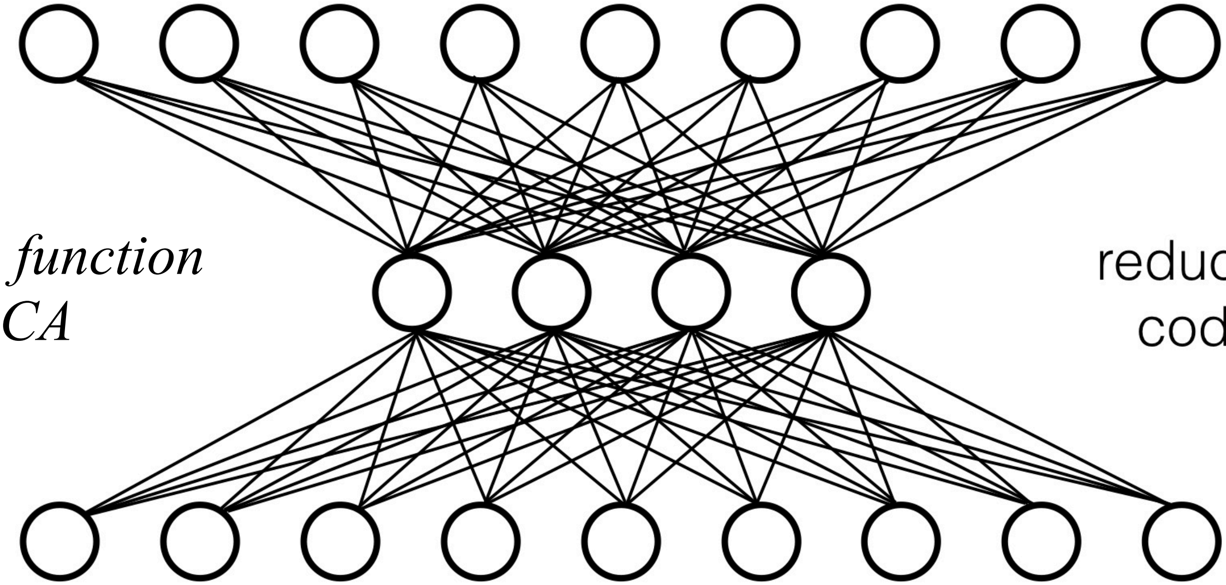
Homework 7 (due Fri Nov 11)

Q1a (5 points). Create an autoassociator network that produces a lower-dimensional representation of the Fashion-MNIST images on its hidden layer. It will use a “flattened” (one-dimensional) version of these images ...

train with 1/2 of the Fashion-MNIST images

reproduce input

linear activation function
"regression problem"



linear activation function
to mimic PCA

reduced
code

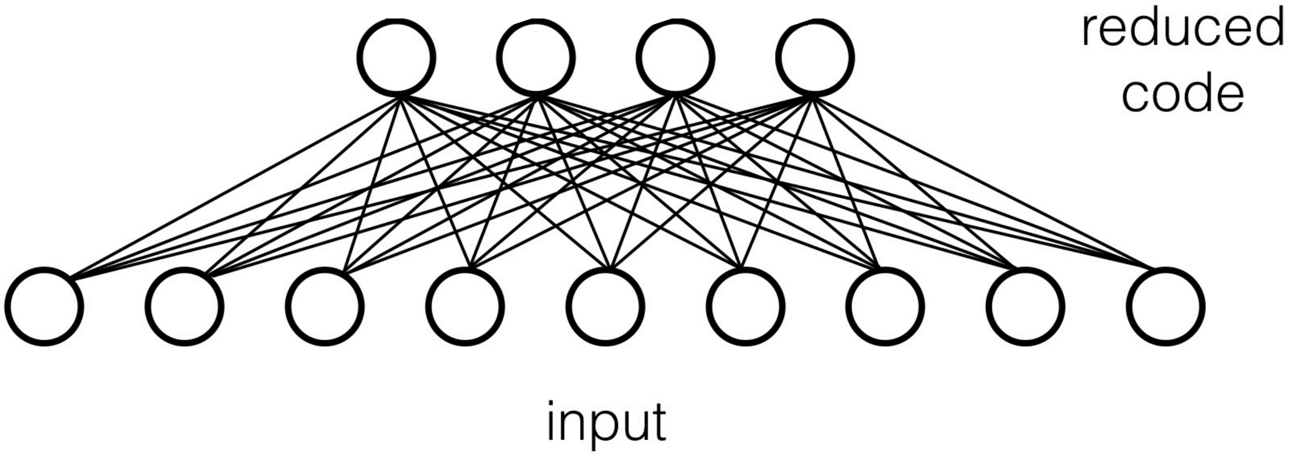
input

need to "flatten" to 1D
like earlier assignments

Homework 7 (due Thu Nov 11)

... After you train your autoassociator, you will need to create a reduced-dimensionality versions of the remaining 1/2 of the Fashion-MNIST training set. ...

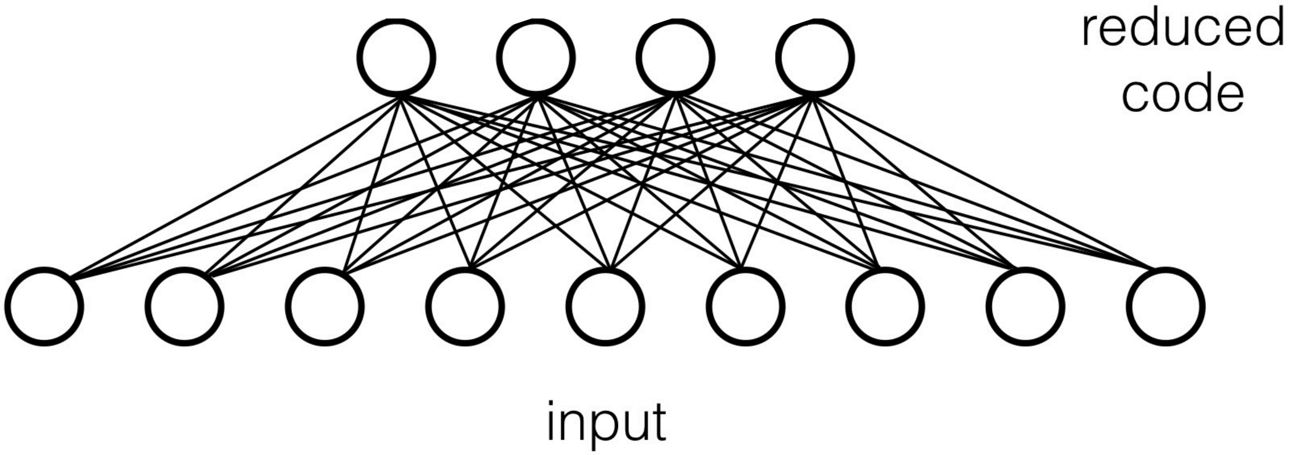
remaining 1/2 of the Fashion-MNIST images



Homework 7 (due Thu Nov 11)

... After you train your autoassociator, you will need to create a reduced-dimensionality versions of the remaining 1/2 of the Fashion-MNIST training set. ...
... The easiest way to do this is to use the weights and biases from the trained autoassociator to calculate the activation values on the hidden layer

```
# get weights and biases from the aa network
aaW = aanetwork.layers[0].get_weights()[0]
aaB = aanetwork.layers[0].get_weights()[1]
```



Homework 7 (due Thu Nov 11)

Q1b (3 points). Now, use these lower-dimensional (PCA-like) representations of the clothing images (remaining 1/2 of the original set) as the inputs to train a densely-connected multi-layered neural network that learns, via backpropagation, to classify the images as one of the 10 types of clothing.

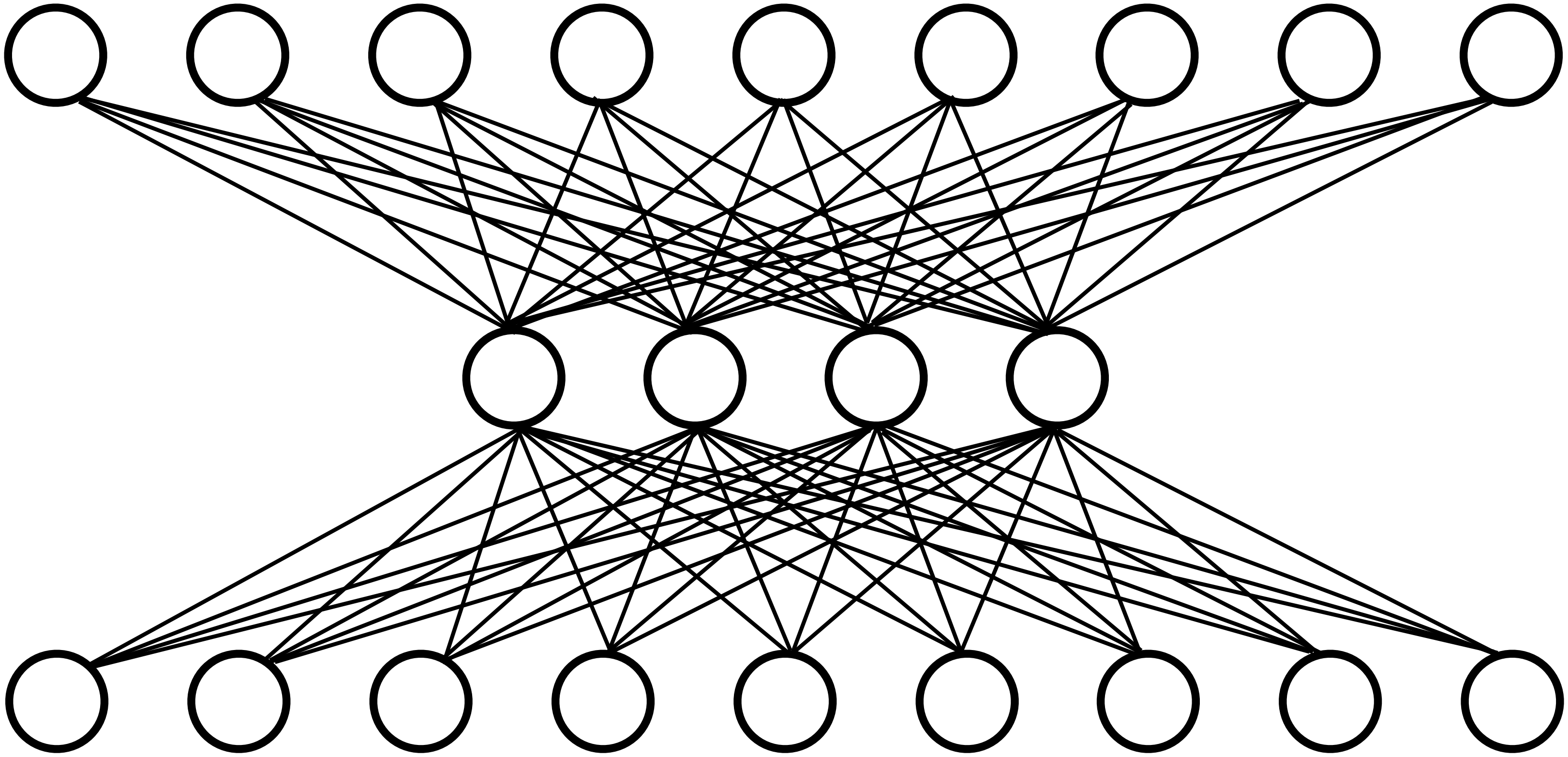
```
# get weights and biases from the aa network
aaW = aanetwork.layers[0].get_weights()[0]
aaB = aanetwork.layers[0].get_weights()[1]
```

using hidden layer activations from the autoassociator
as input to a multi-layer classification network

autoassociator

from Week10a

once the reduced code is learned,
that code can be used as input to a backprop network



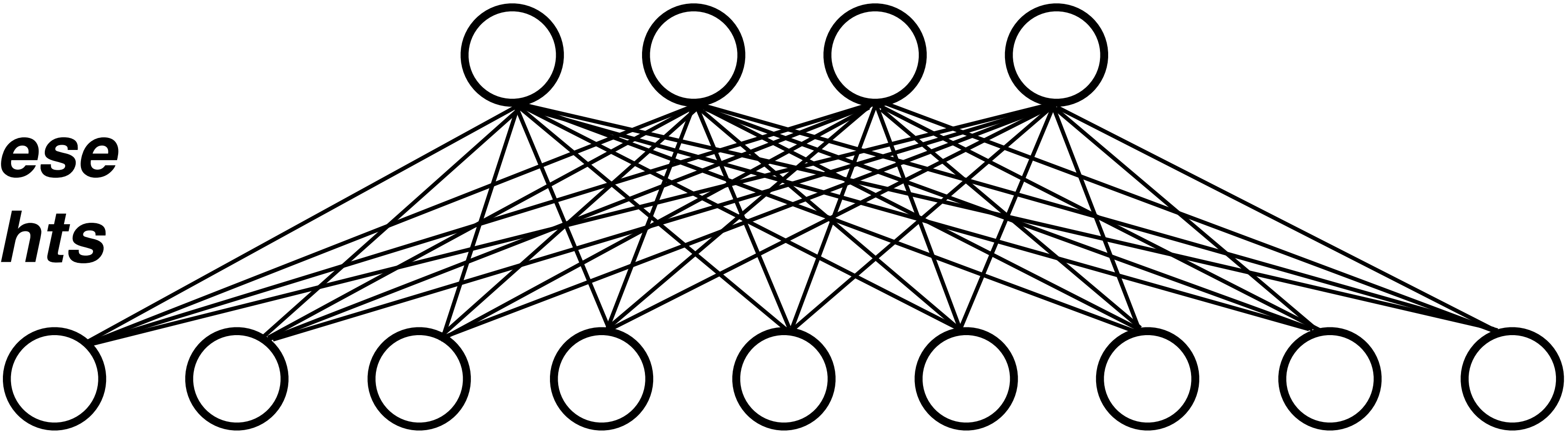
input

autoassociator

from Week10a

(instead of "fixing these weights", just use the weights and biases to calculate the activations, which will then be used as inputs to a trained network)

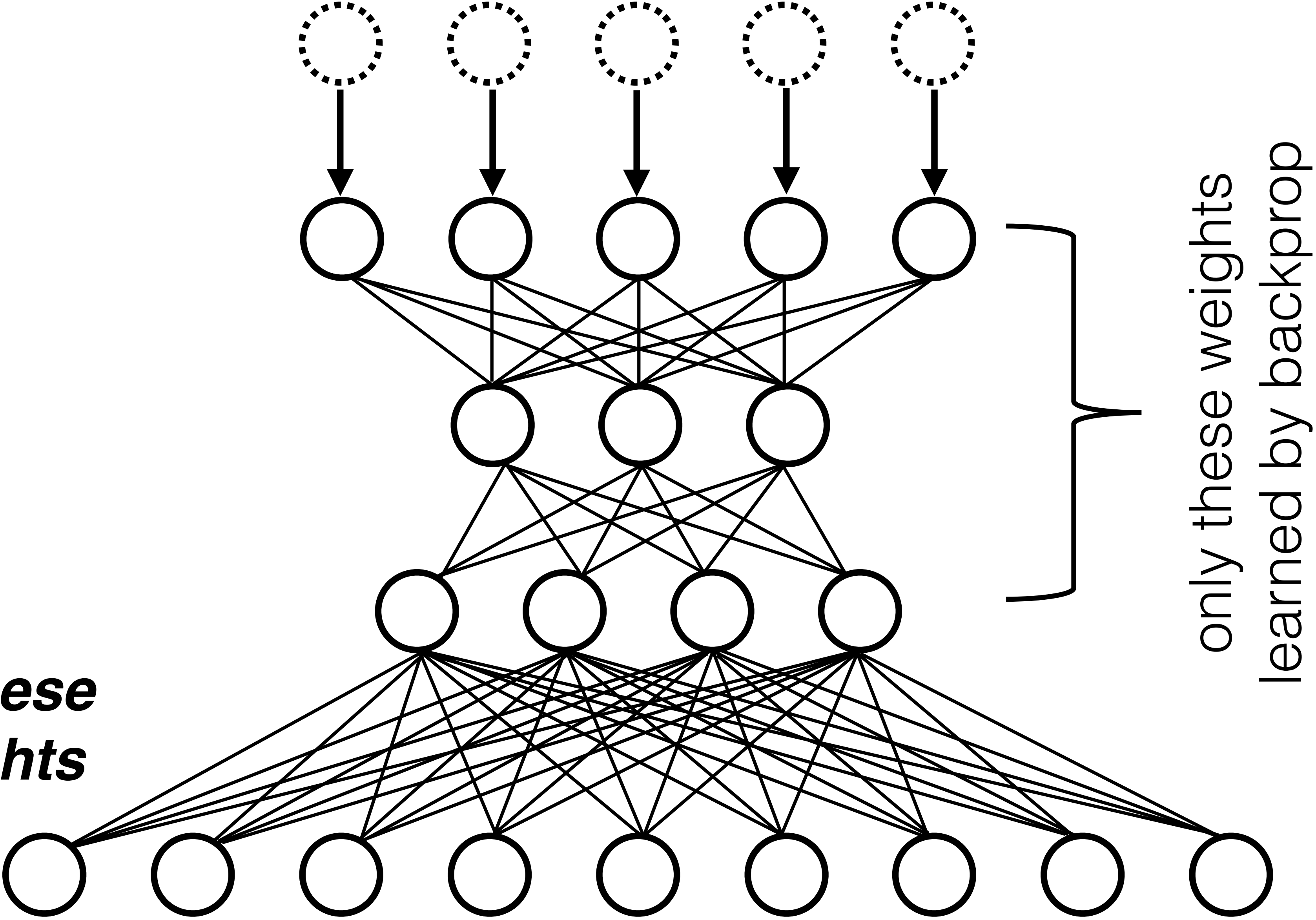
fix these weights



input

use autoassociator representations in feedforward net

from Week10a



fix these weights

only these weights learned by backprop

input

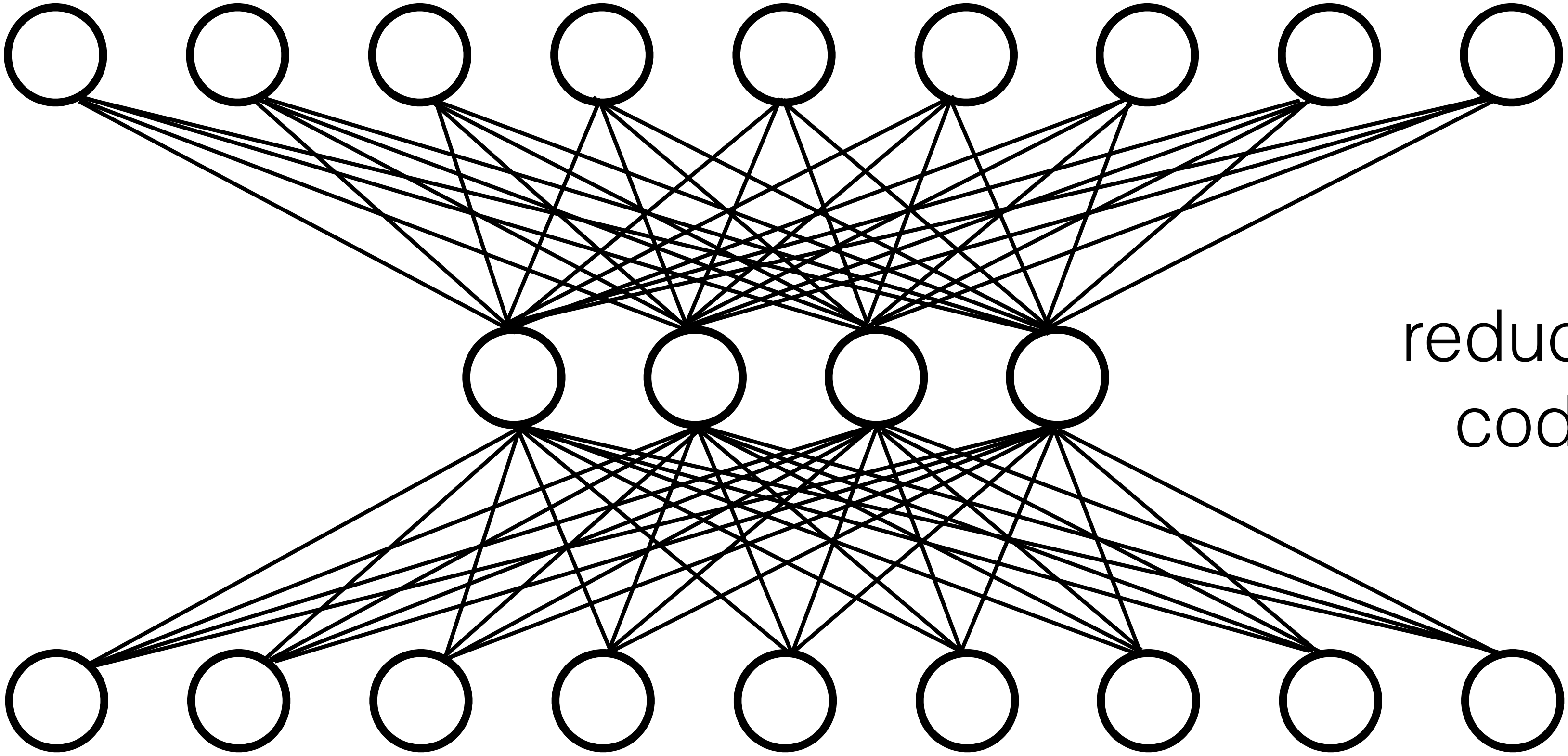
autoassociators and principal components analysis (PCA)

type of neural network

statistical technique

autoassociator

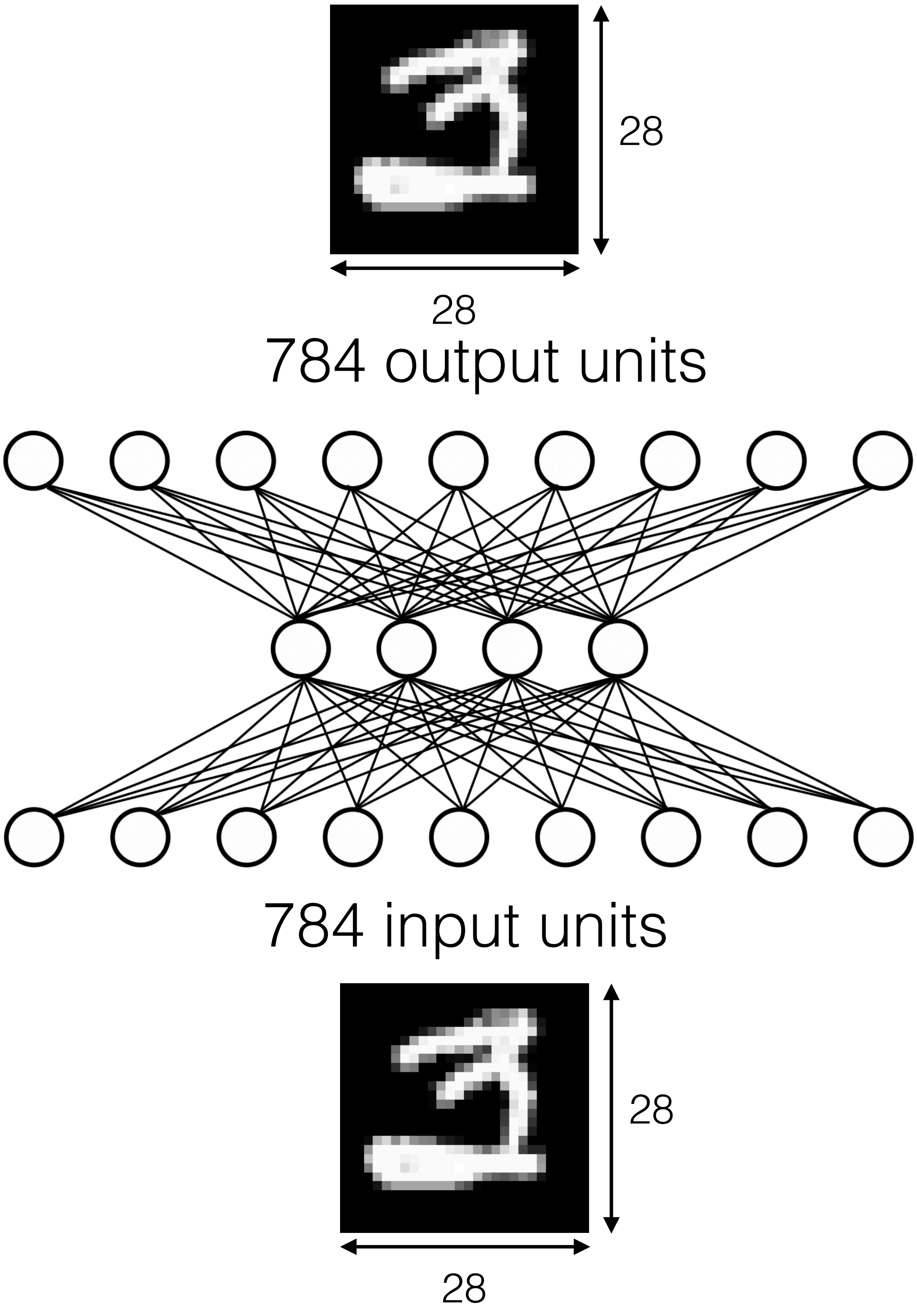
using supervised learning to do unsupervised learning



reproduce input

reduced code

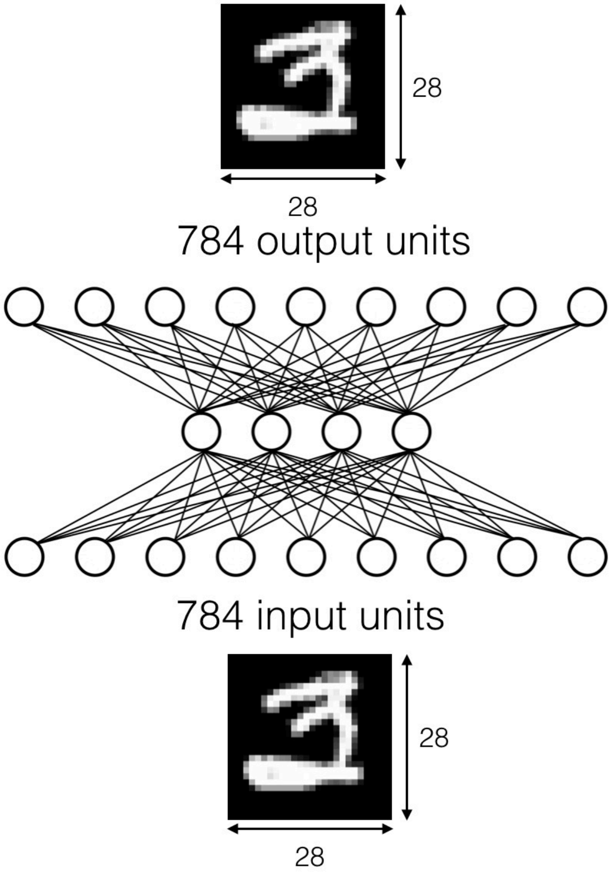
input



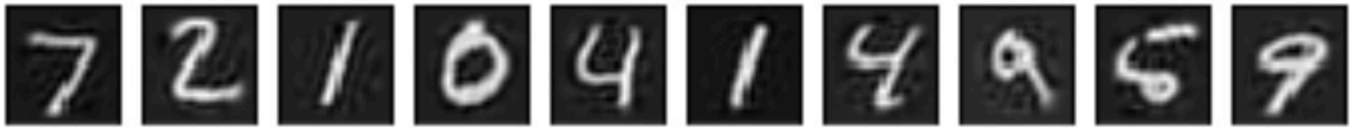
784 output units

784 input units

represent images in
784-dimensional space
using fewer
hidden unit dimensions

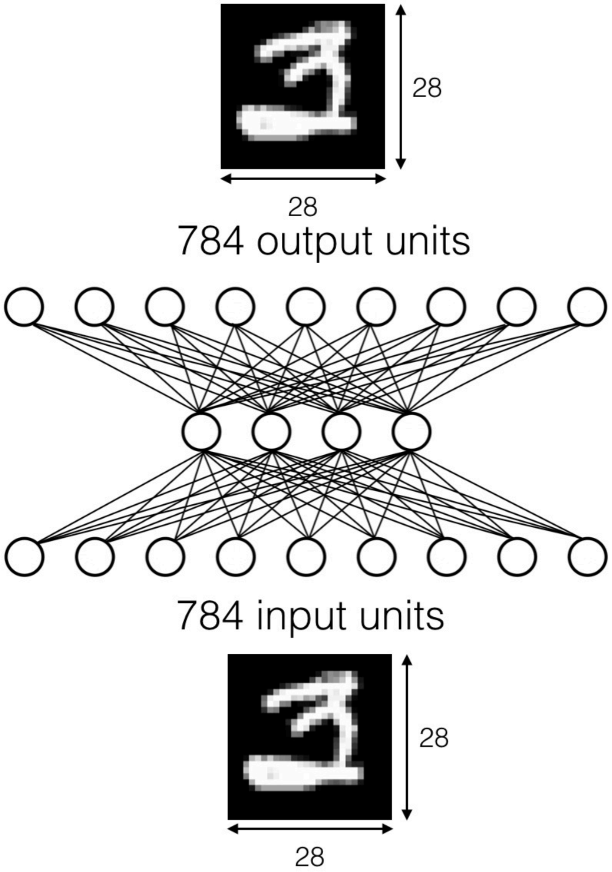


output images



with 100 hidden units

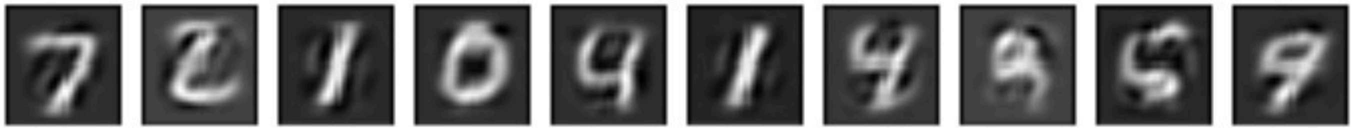
input images



784 output units

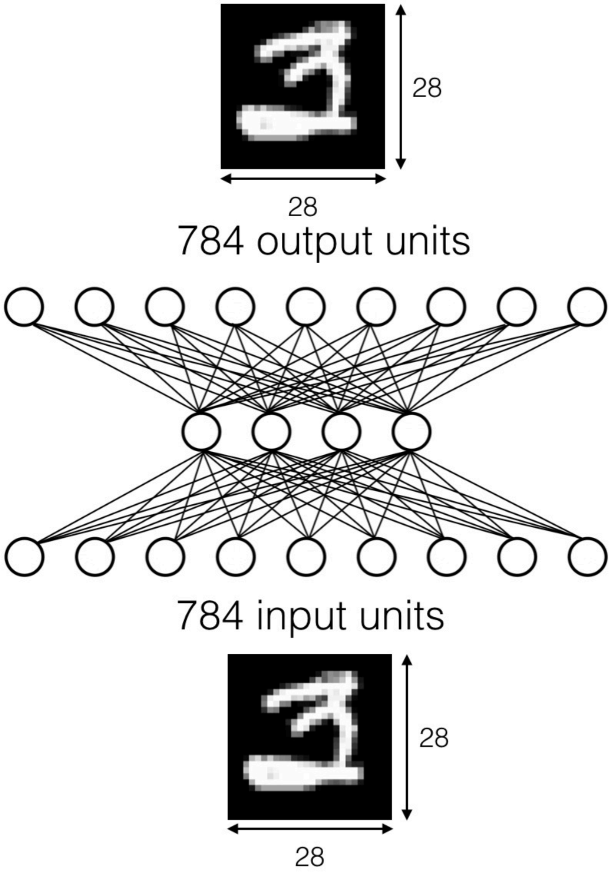
784 input units

output images

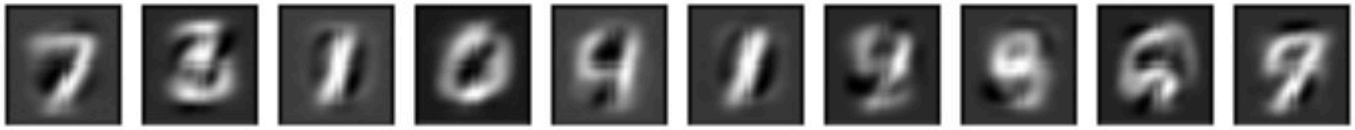


with 25 hidden units

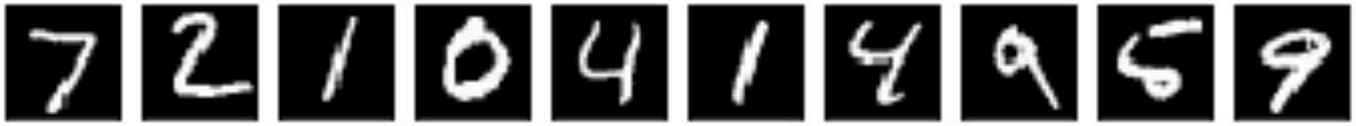
input images



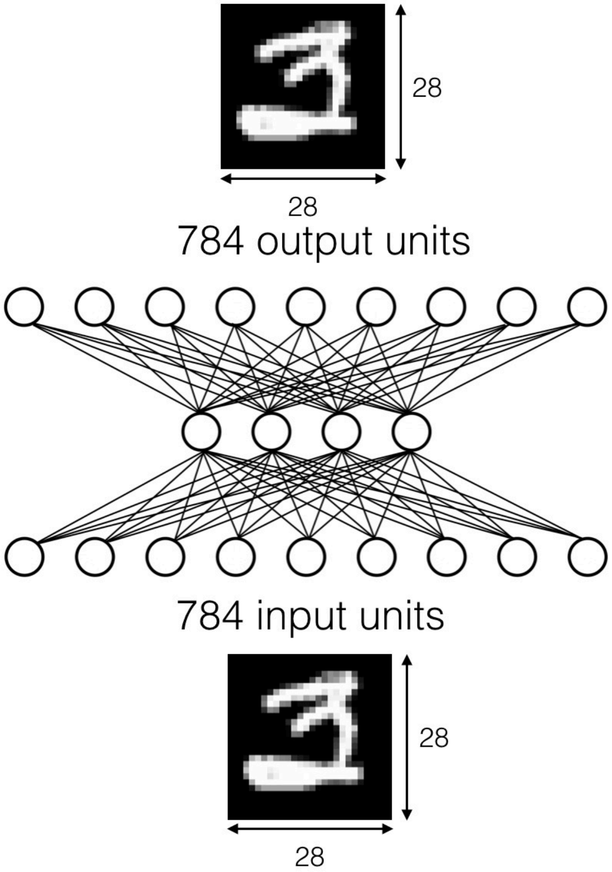
output images



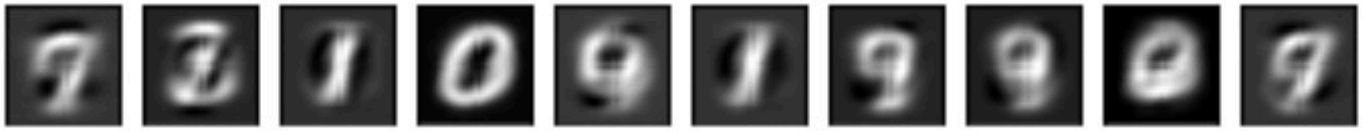
with 10 hidden units



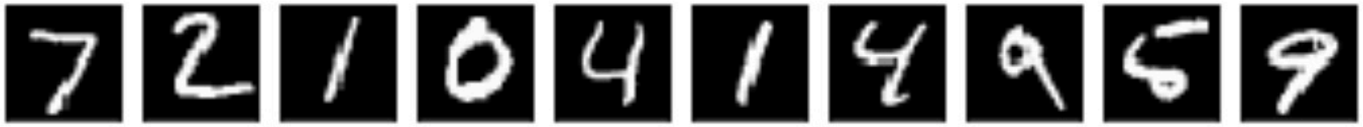
input images



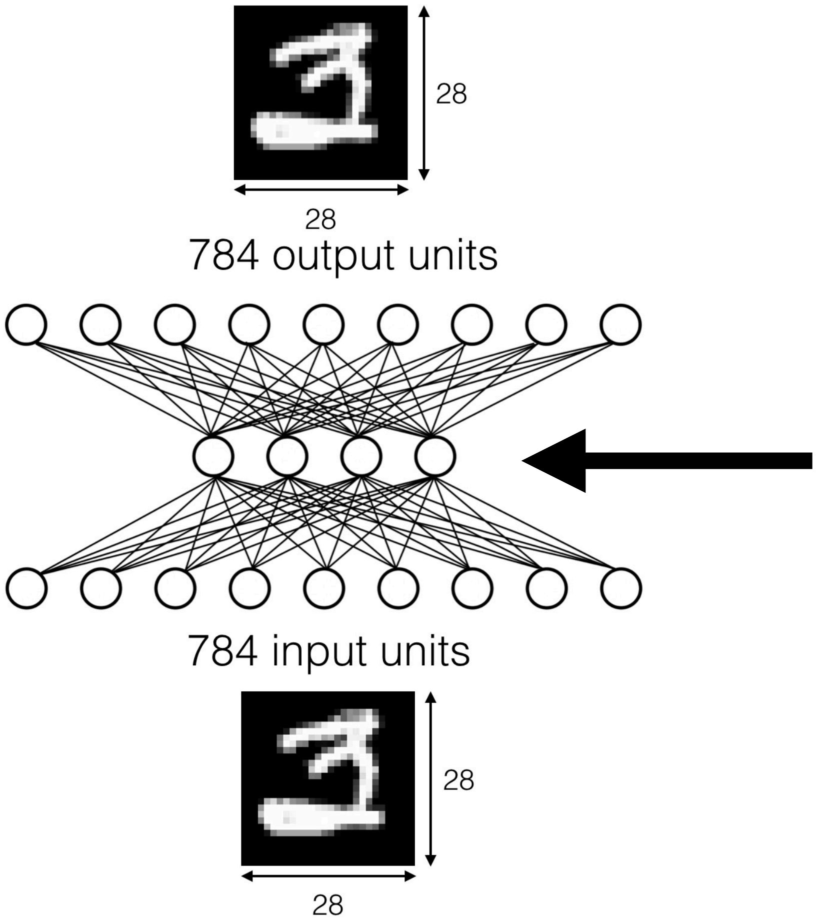
output images



with 5 hidden units



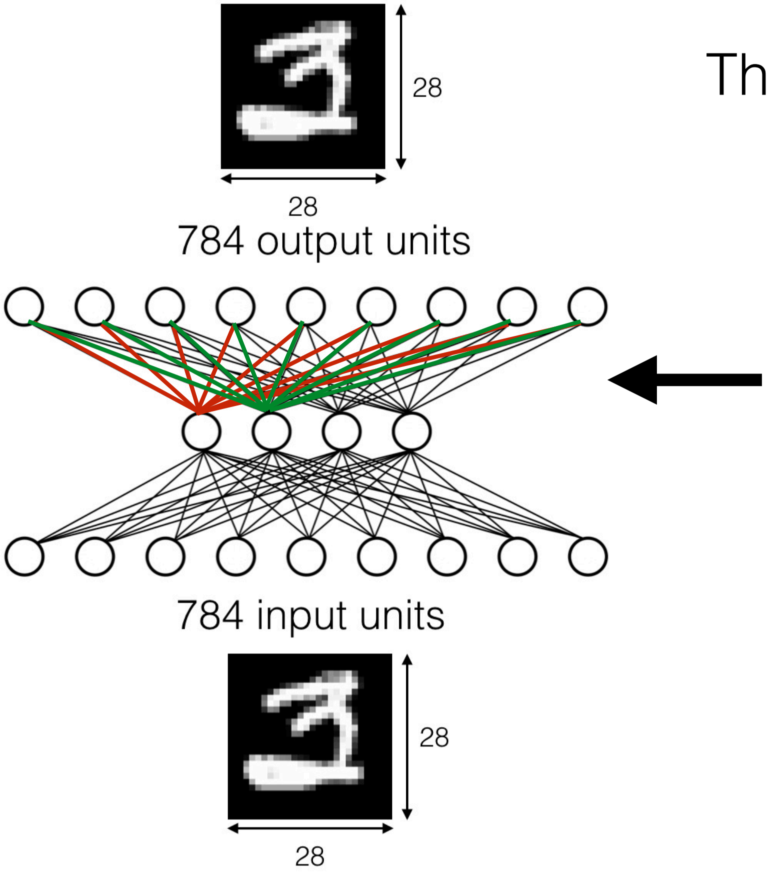
input images



What kinds of representations does the autoassociator learn?

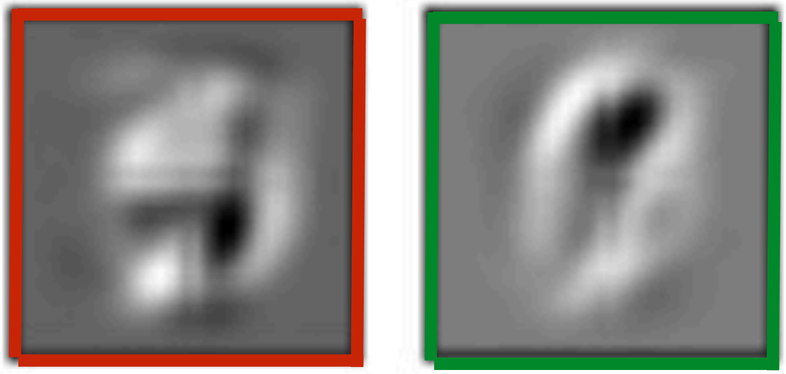
Does it try to learn the “parts” that make up different letters?

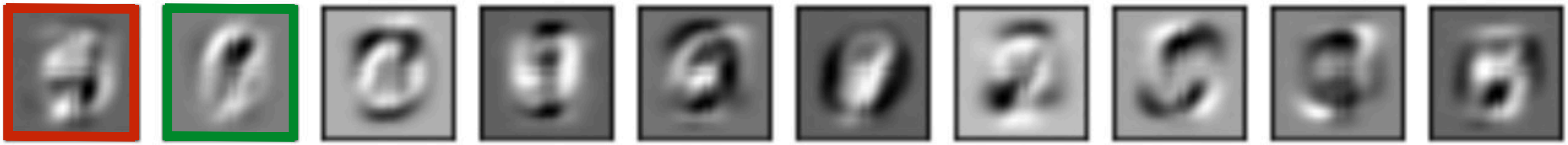
The autoassociator learns “holistic” building blocks.



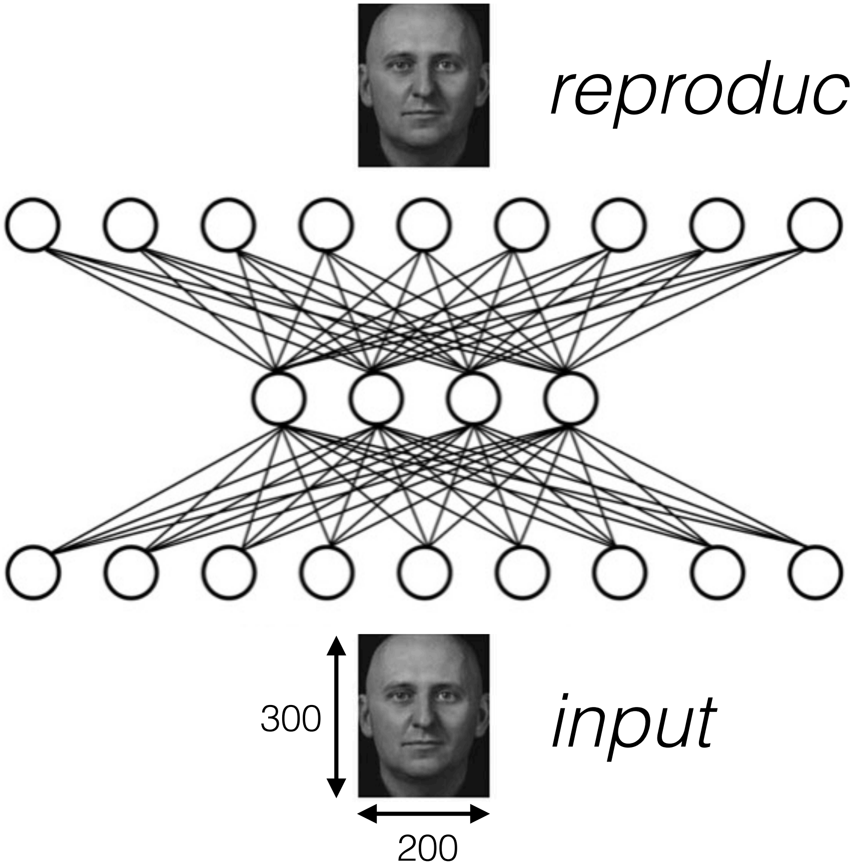
assuming 10 hidden units

images produced by weights



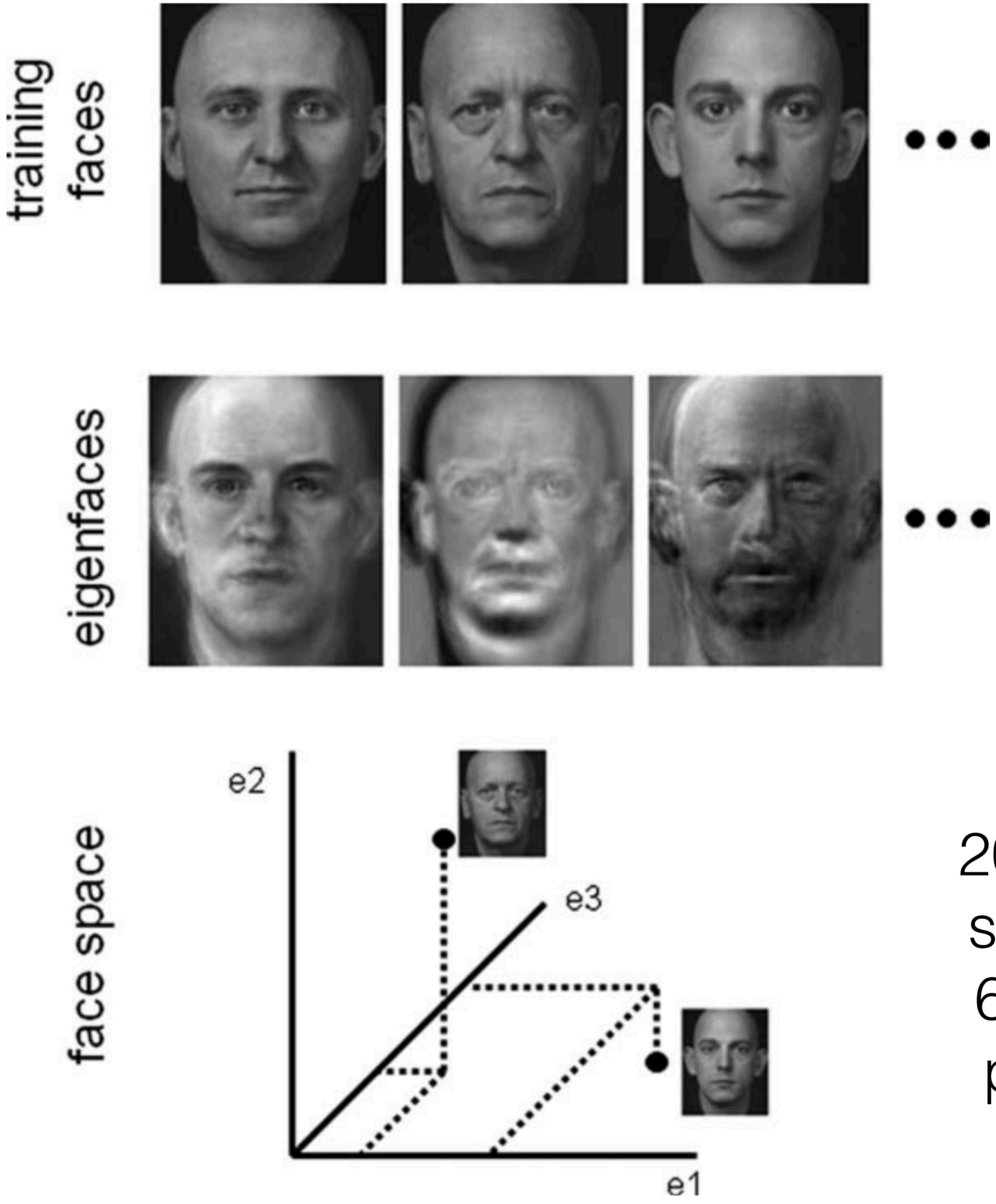


training faces



reproduced

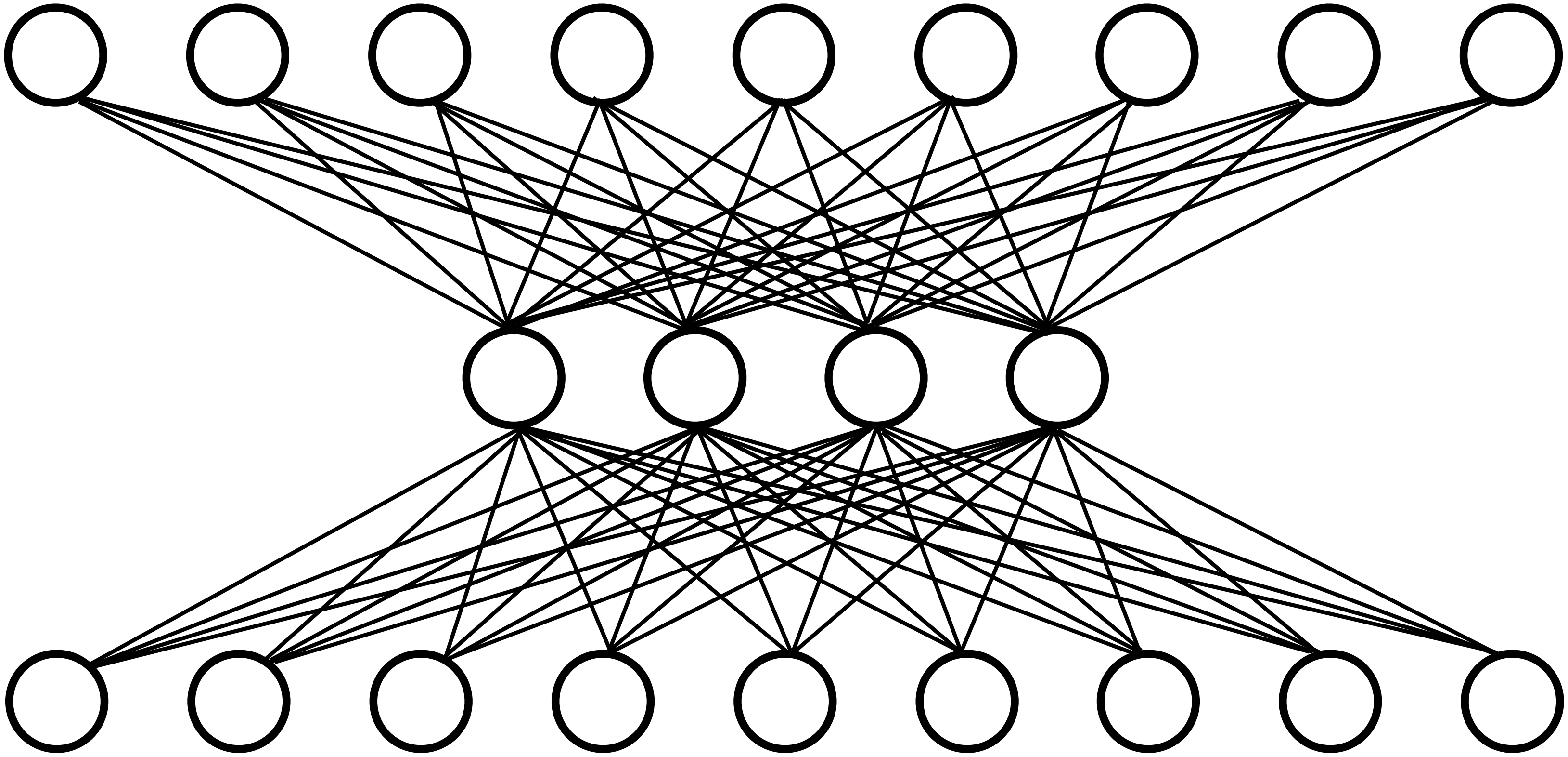
input



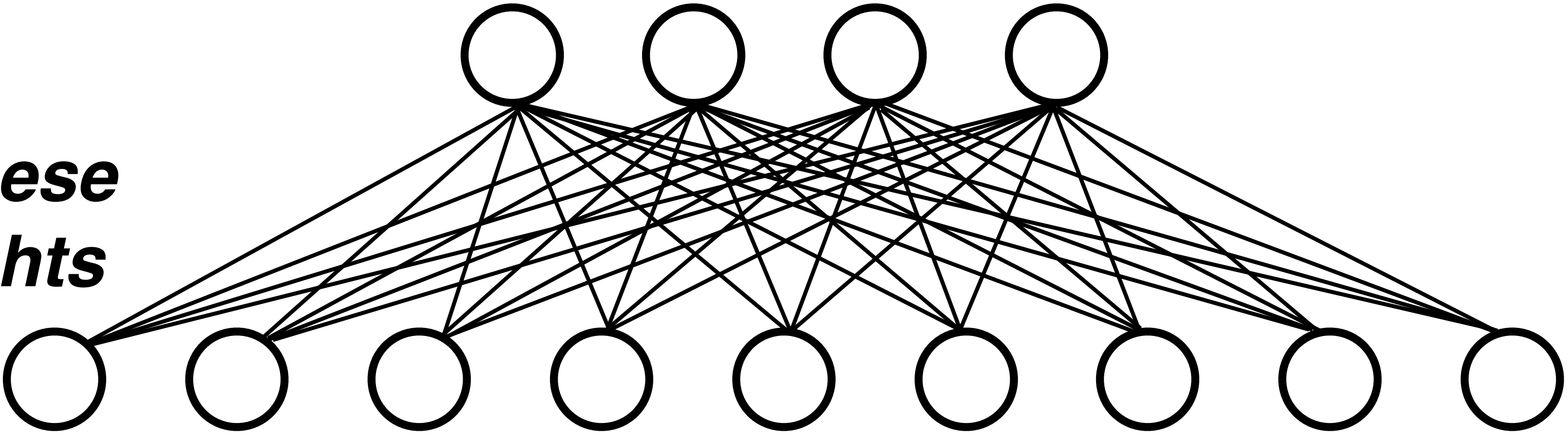
20-dimensional face space rather than a 60000-dimensional pixel image space

autoassociator

once the reduced code is learned,
that code can be used as input to a backprop network



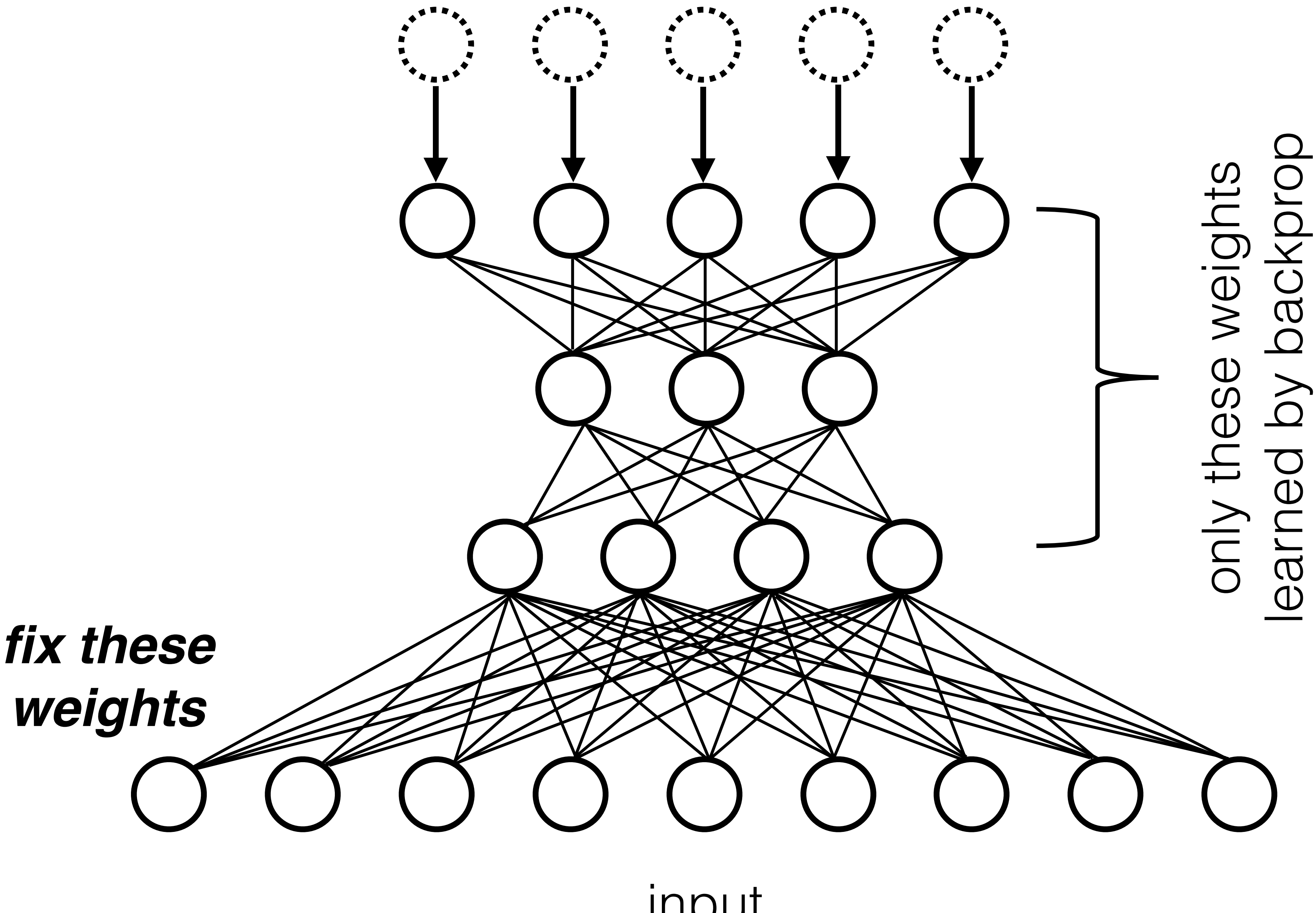
autoassociator



fix these weights

input

use autoassociator representations in feedforward net



Homework 7 (due Thu Nov 11)

Q2 (10 points). Create a convolutional neural network that classifies the Fashion-MNIST images. ...

(how to do this will be described today and Thu)

Programming Deep Convolutional Neural Networks using Keras / Tensorflow

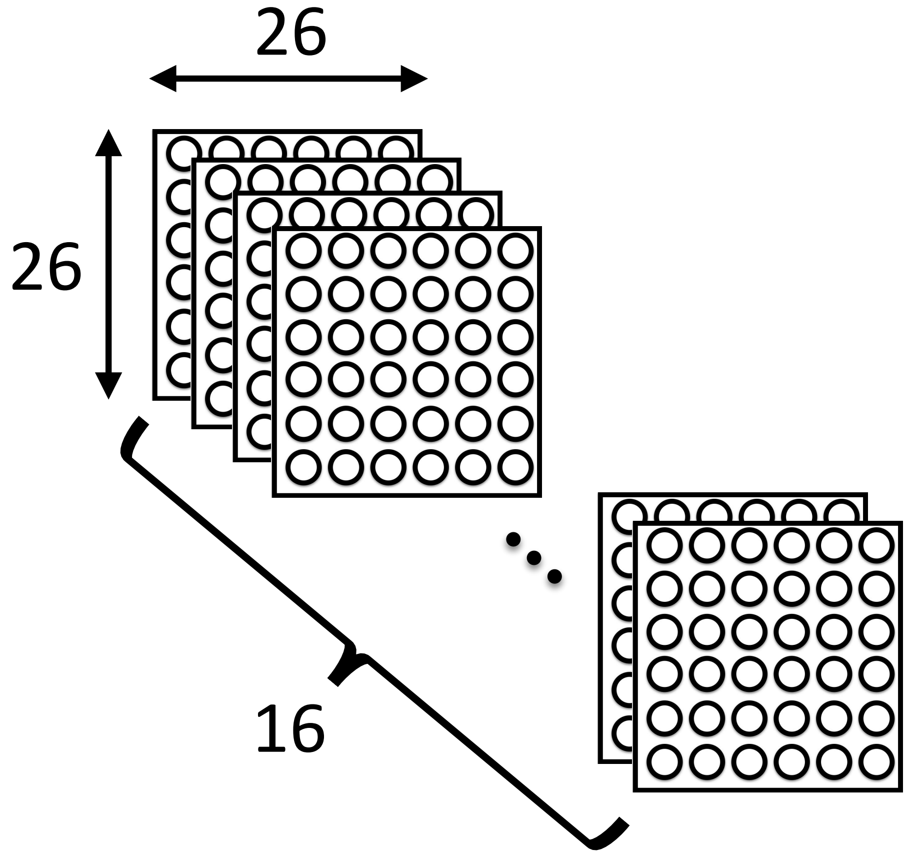
```
# this initializes a blank Sequential network
network = models.Sequential()
```

```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

layers.Conv2D is a type of layer
(before we used **layers.Dense**)

```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

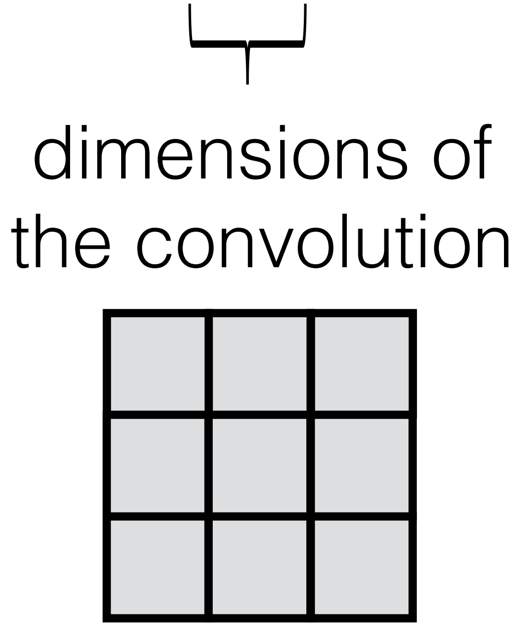
number of feature maps



```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))

for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                             format(layer.name,   layer.input.shape, layer.output.shape))
```

```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 26, 26, 16)
```

```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```



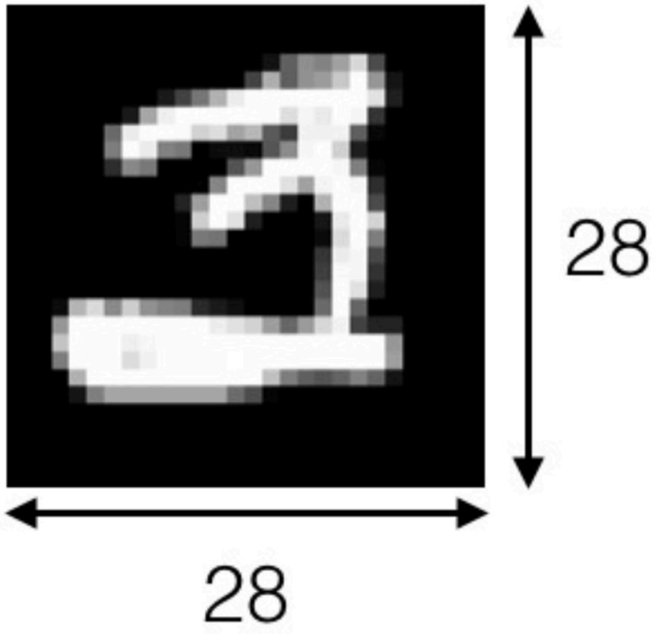
dimensions of
the convolution

```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

activation
function

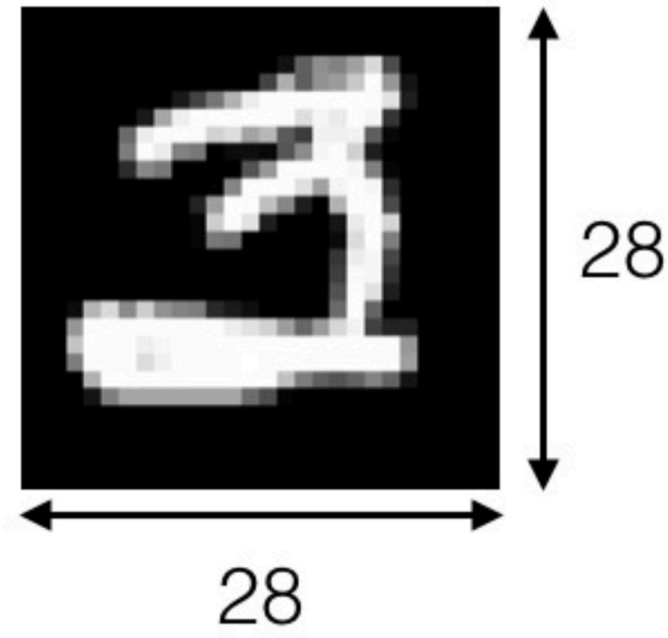
```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

first layer requires
input shape



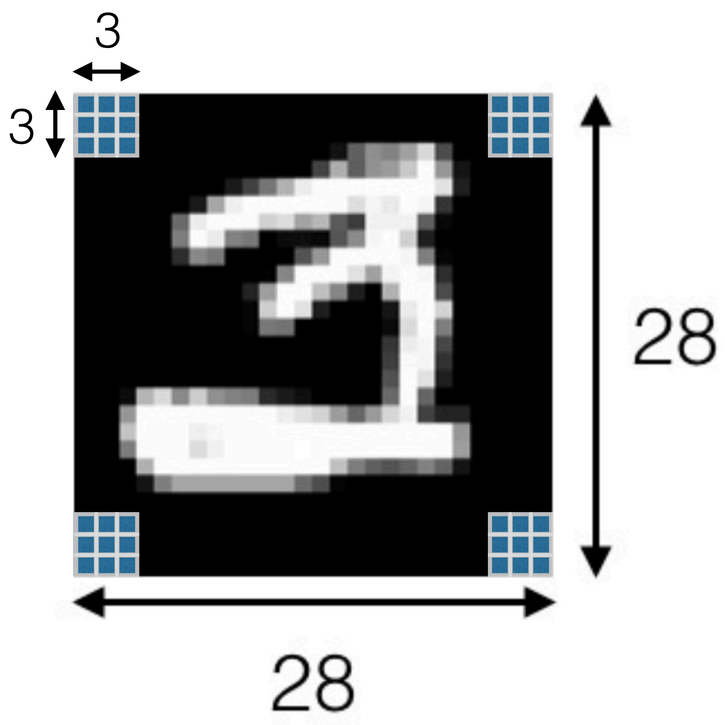
```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))

for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                        format(layer.name,   layer.input.shape, layer.output.shape))
```

```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 26, 26, 16)
```



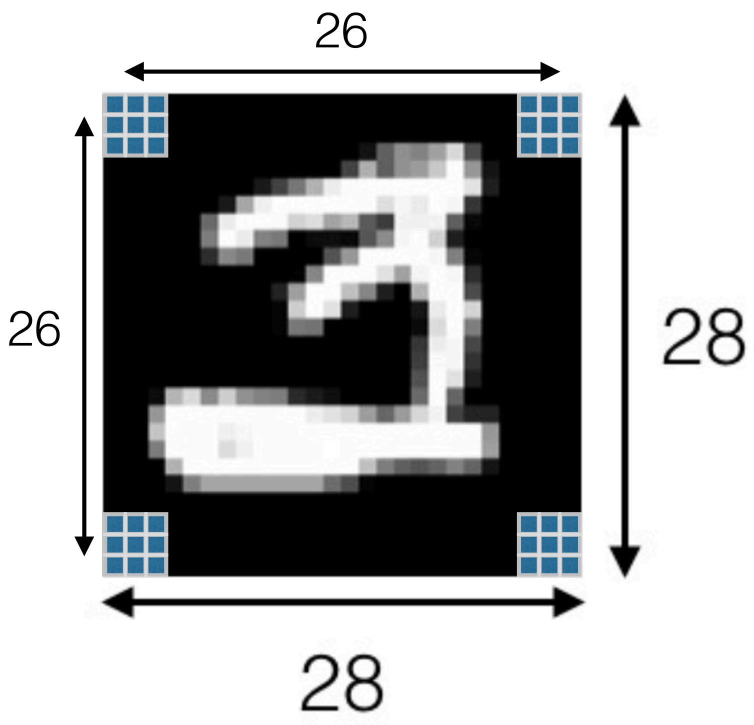
```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))

for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                  format(layer.name,   layer.input.shape, layer.output.shape))
```

```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 26, 26, 16)
```



```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))

for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                                format(layer.name,   layer.input.shape, layer.output.shape))
```

```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 26, 26, 16)
```



```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

required with Conv2D layer

what is this third dimension?

```
# load fashion mnist dataset
from tensorflow.keras.datasets import fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

remember that we reshape this from a matrix to a vector when training Dense networks

now we’re keeping the 2D structure, but …

```
# load fashion mnist dataset
from tensorflow.keras.datasets import fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

print(train_images.shape)
```

```
(60000, 28, 28)
```

```
# load fashion mnist dataset
from tensorflow.keras.datasets import fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

print(train_images.shape)
(60000, 28, 28)
```

error without
this

```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))

for layer in network.layers:
   print('layer name : {} | input shape : {} | output shape : {}'.
                                    format(layer.name,   layer.input.shape, layer.output.shape))

layer name : conv2d_1 | input shape : (?, 28, 28, 1) | output shape : (?, 26, 26, 16)
```

```
# load fashion mnist dataset
from tensorflow.keras.datasets import fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

print(train_images.shape)
(60000, 28, 28)

network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))

for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                             format(layer.name,   layer.input.shape, layer.output.shape))

layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 26, 26, 16)
```

need to reshape the train_images (and test_images)

```
train_images = train_images.reshape((60000, 28, 28, 1))
```

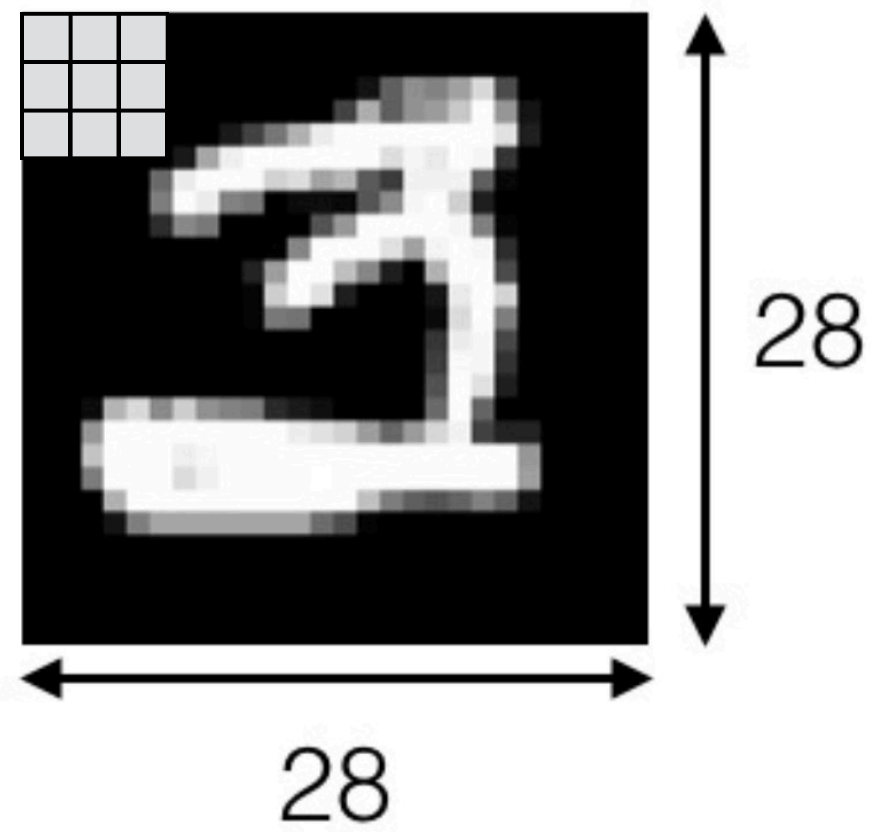
(use variables rather than hard-code)

```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

, padding='valid',

optional parameter
default is **'valid'**

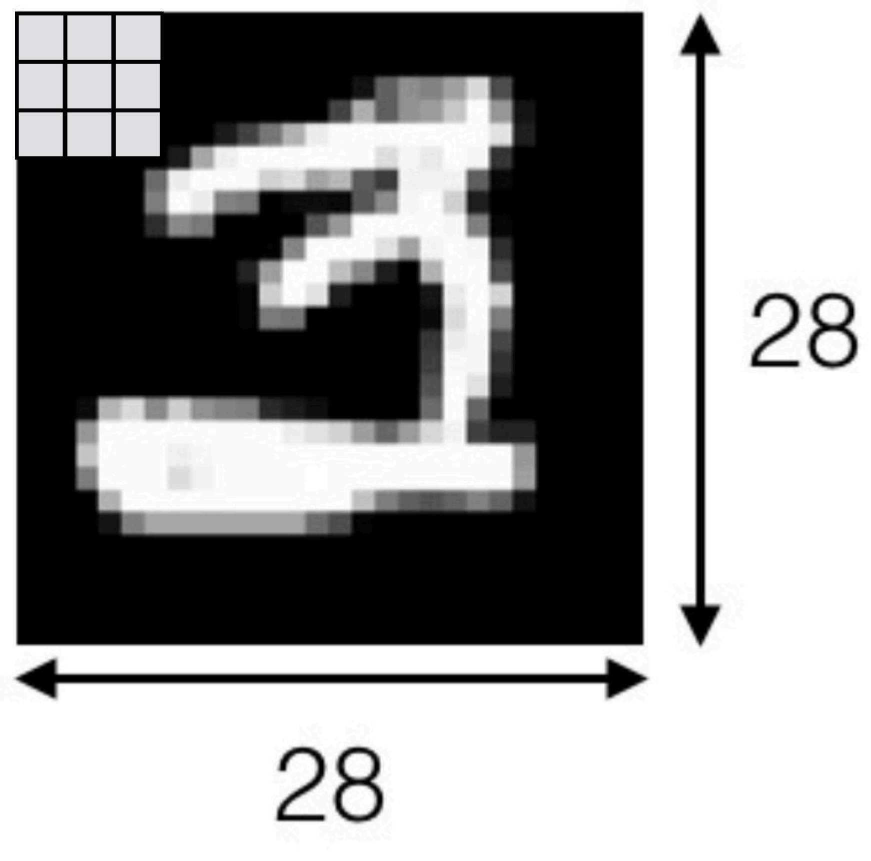
with **'valid'** it stays
inside previous layer



```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

what will be the dimensions of the Conv2D layer?

with **'valid'** it stays inside previous layer



```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))

for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                    format(layer.name,   layer.input.shape, layer.output.shape))
```

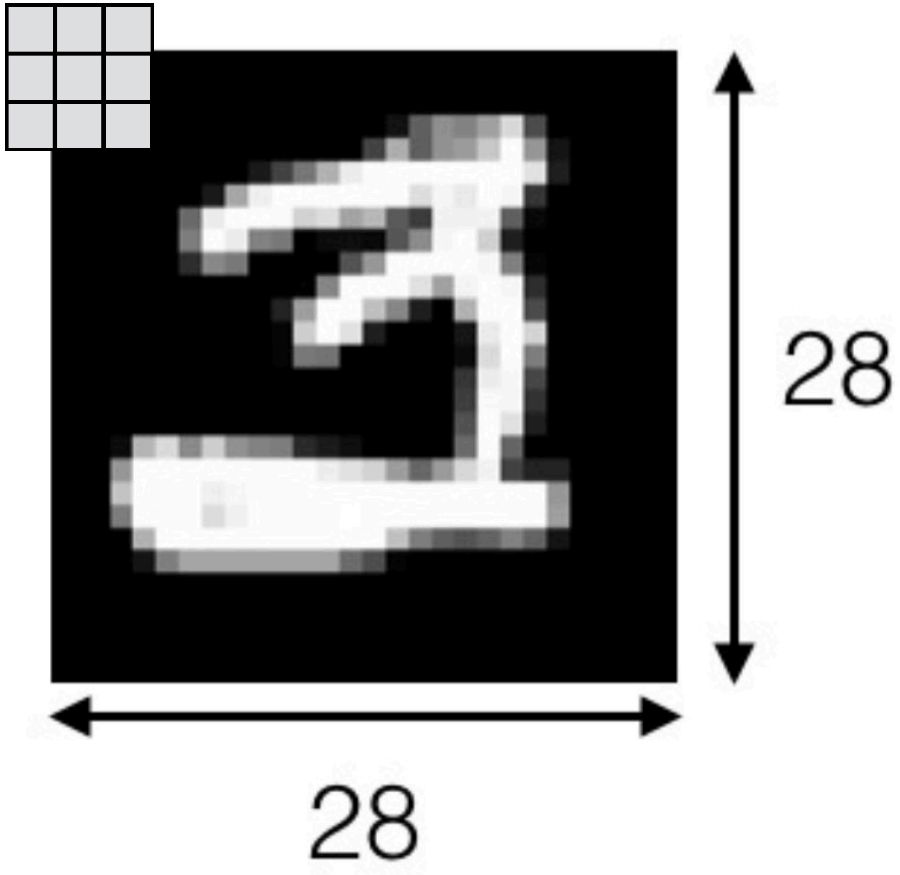
```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 26, 26, 16)
```

```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

may want to use '`same`'
in Homework 7

```
, padding='same',
```

optional parameter
default is **'valid'**



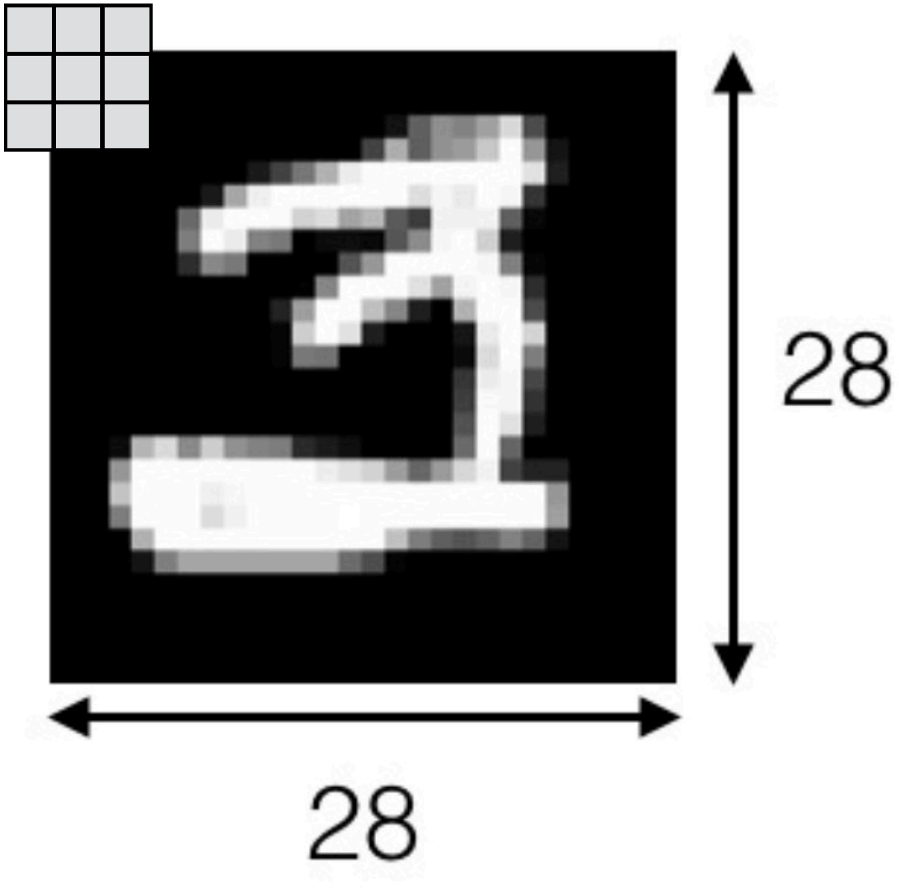
with **'same'** it goes
outside previous layer

```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

may want to use 'same'
in Homework 7

what will be the dimensions of the Conv2D layer?

with **'same'** it goes
outside previous layer



```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), padding='same', activation='relu',
                              input_shape=(sz,sz,1)))

for layer in network.layers:
   print('layer name : {} | input shape : {} | output shape : {}'.
                                      format(layer.name,   layer.input.shape, layer.output.shape))
```

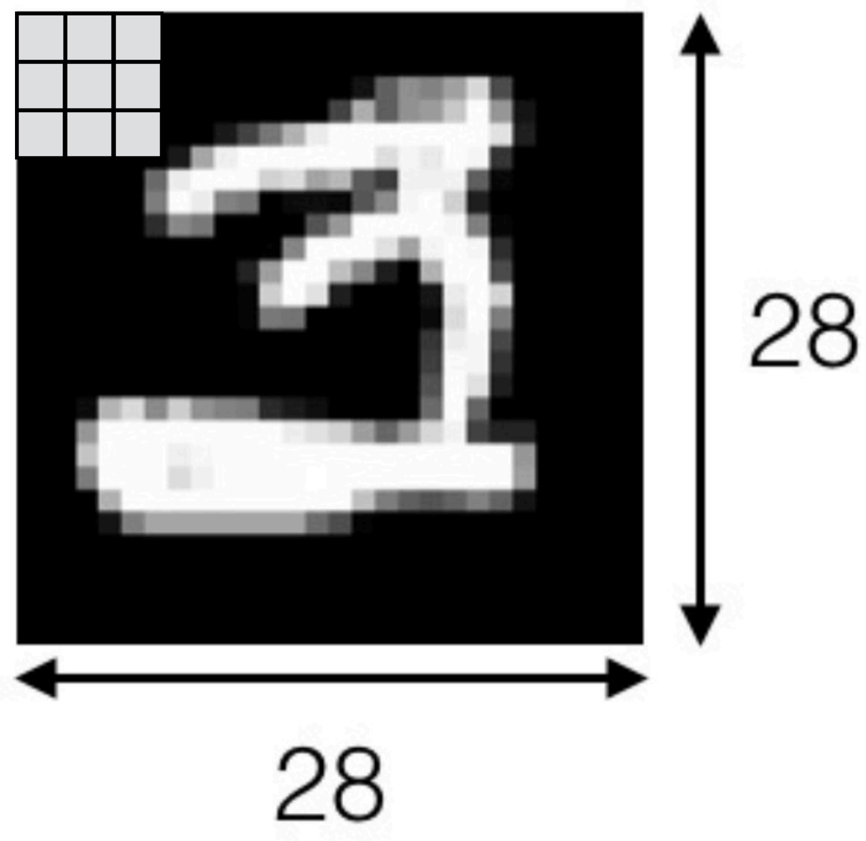
```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 28, 28, 16)
```

```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

, strides=(1, 1),

optional parameter
default is (1,1)

with (1,1) stride,
convolution full covers
previous layer

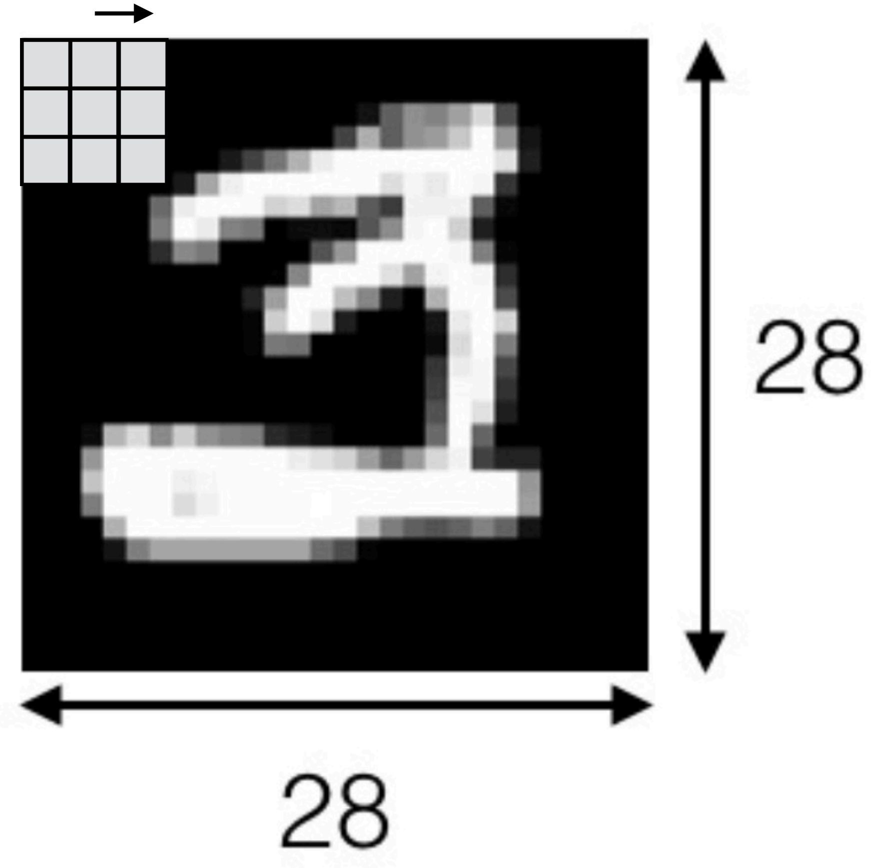


```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

, strides=(1, 1),

optional parameter
default is (1,1)

with (1,1) stride,
convolution full covers
previous layer

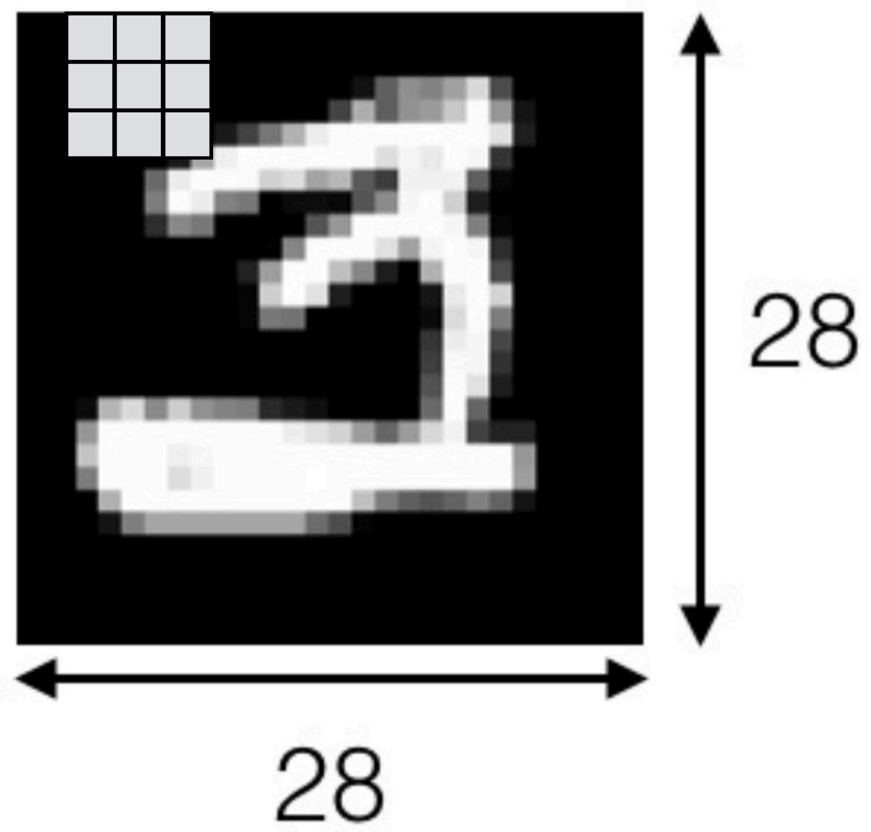


```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

, strides=(1, 1),

optional parameter
default is (1,1)

with (1,1) stride,
convolution full covers
previous layer

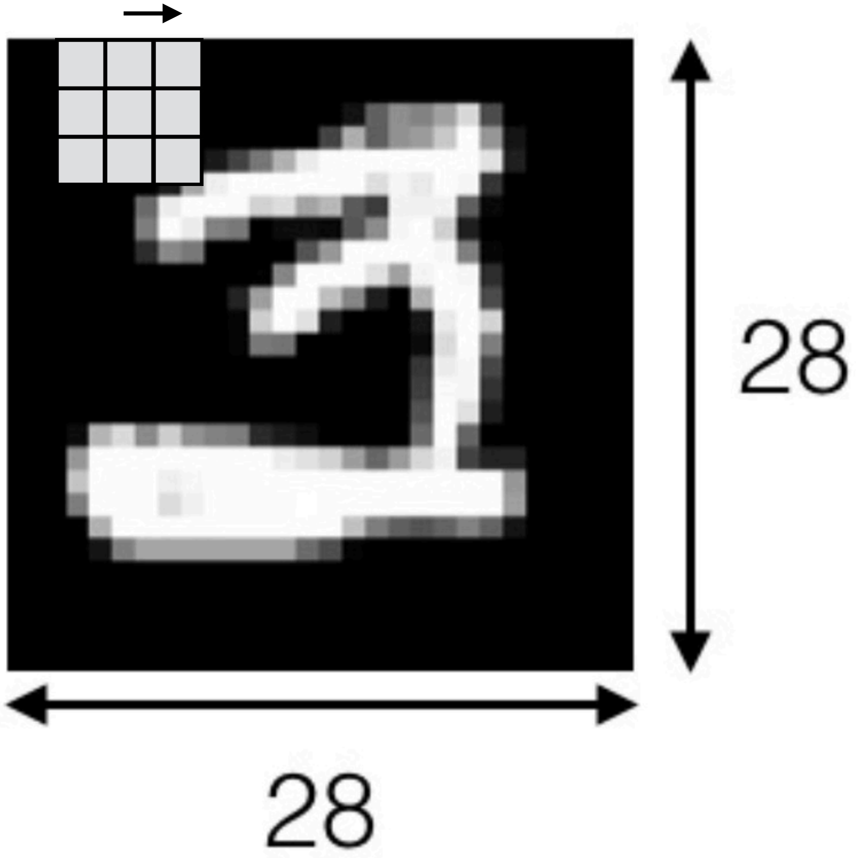


```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

```
, strides=(1, 1),
```

optional parameter
default is (1,1)

with (1,1) stride,
convolution full covers
previous layer



```
# this initializes a blank Sequential network
network = models.Sequential()
```

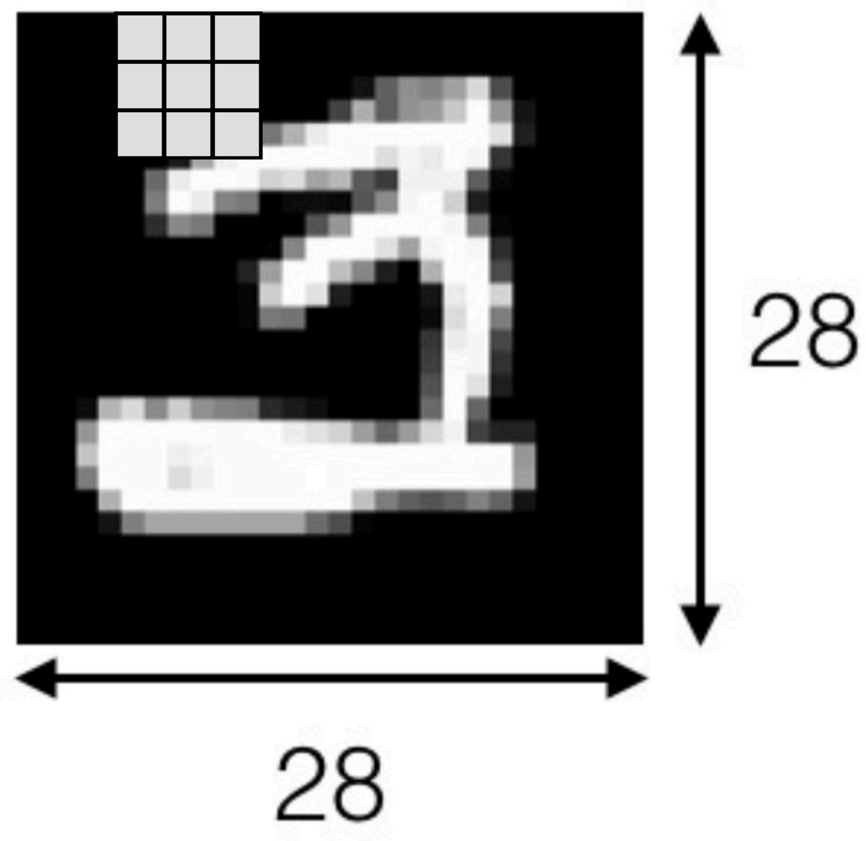
usually use full stride

```
# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

```
, strides=(1, 1),
```

optional parameter
default is (1,1)

with (1,1) stride, convolution full covers previous layer

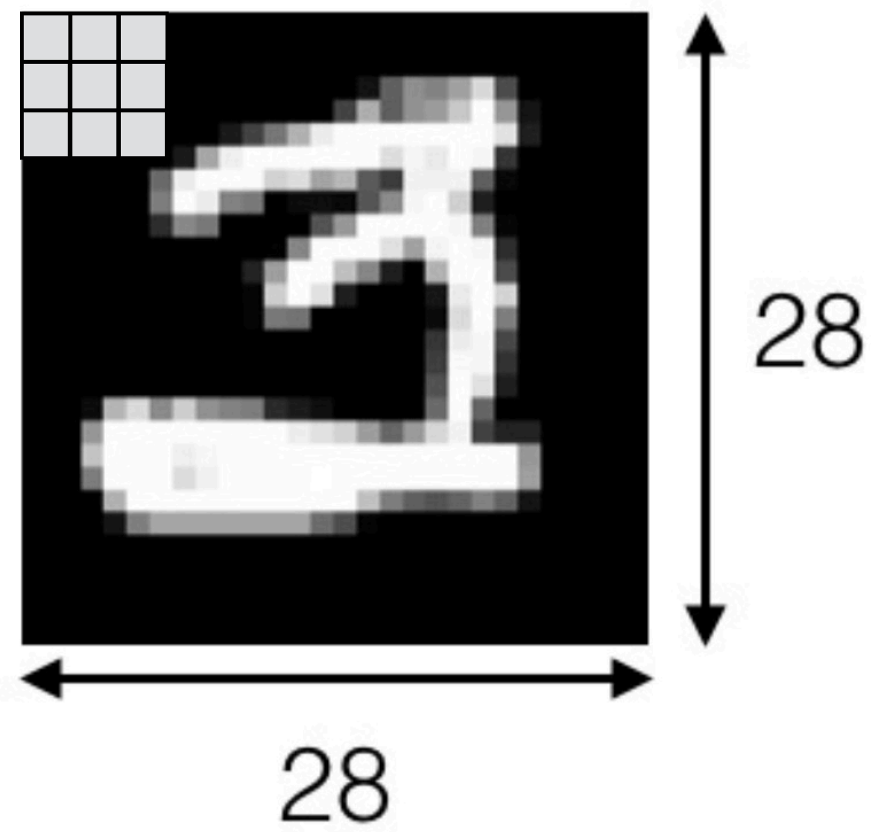


```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

, strides=(2, 2),

optional parameter
default is (1,1)

with (2,2) stride,
convolution has gaps

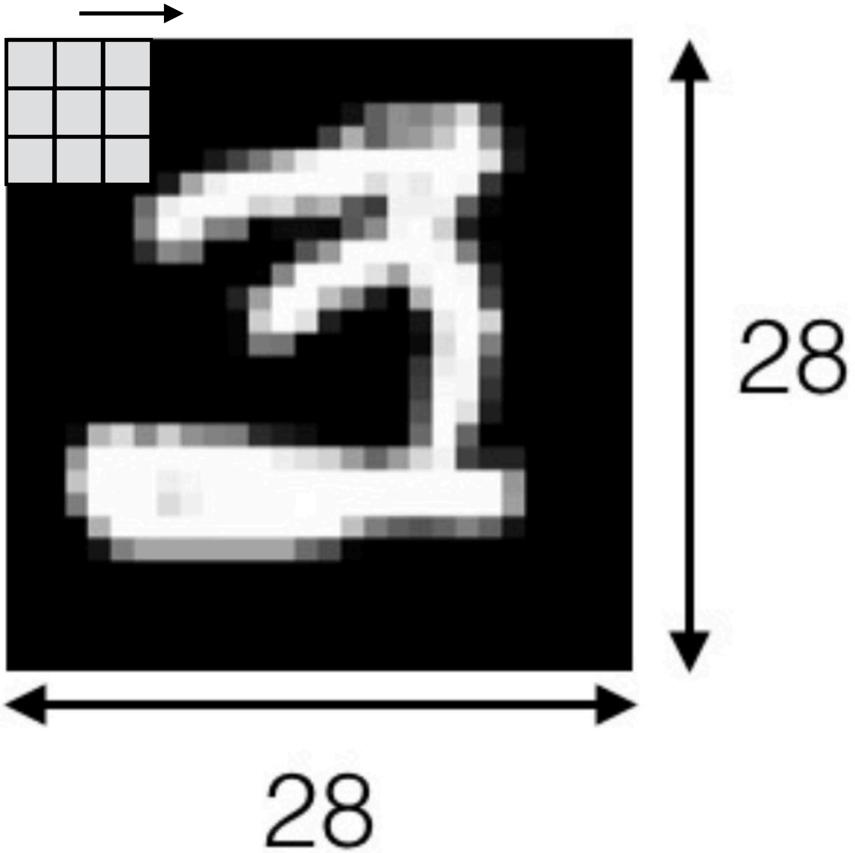


```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

, strides=(2, 2),

optional parameter
default is (1,1)

with (2,2) stride,
convolution has gaps

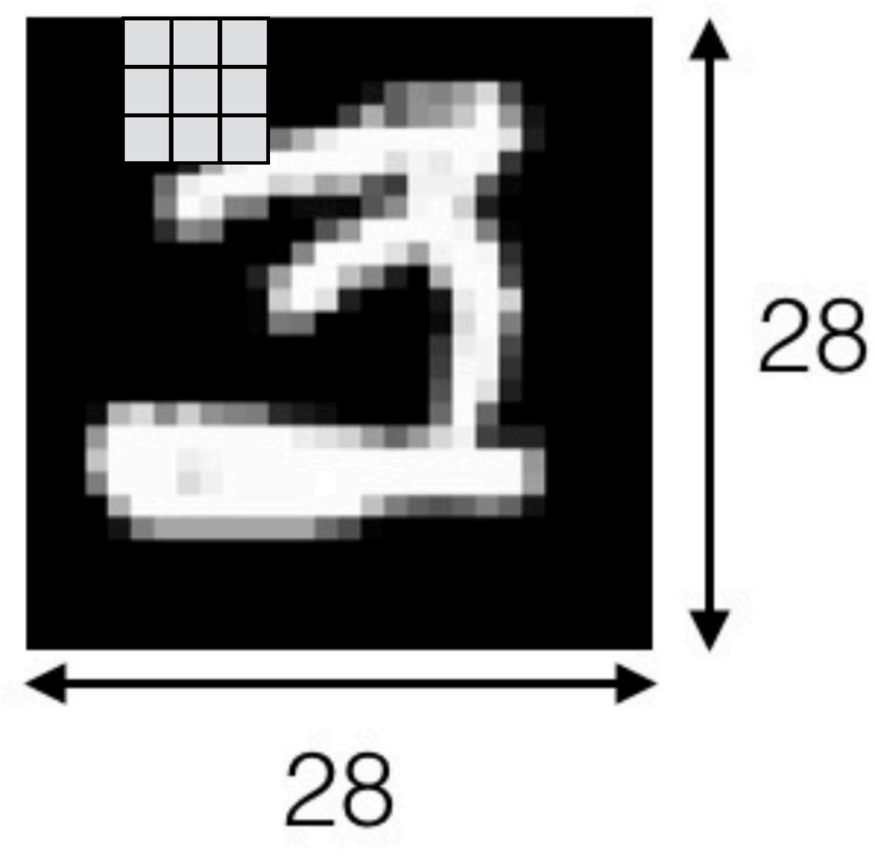


```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

, strides=(2, 2),

optional parameter
default is (1,1)

with (2,2) stride,
convolution has gaps

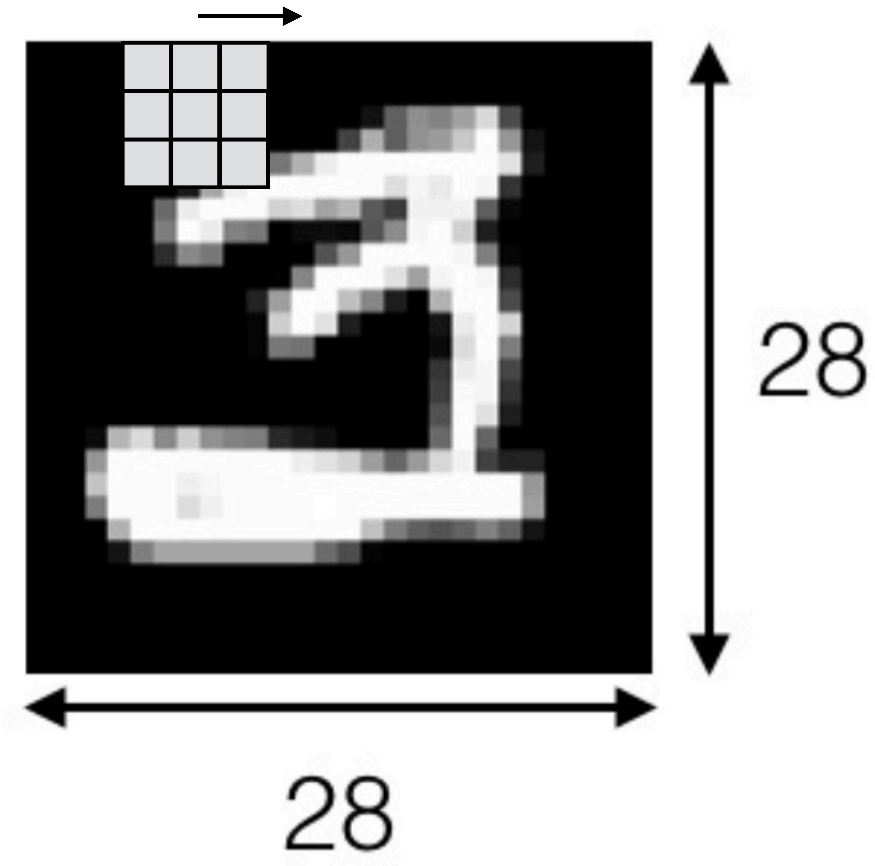


```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

, strides=(2, 2),

optional parameter
default is (1,1)

with (2,2) stride,
convolution has gaps

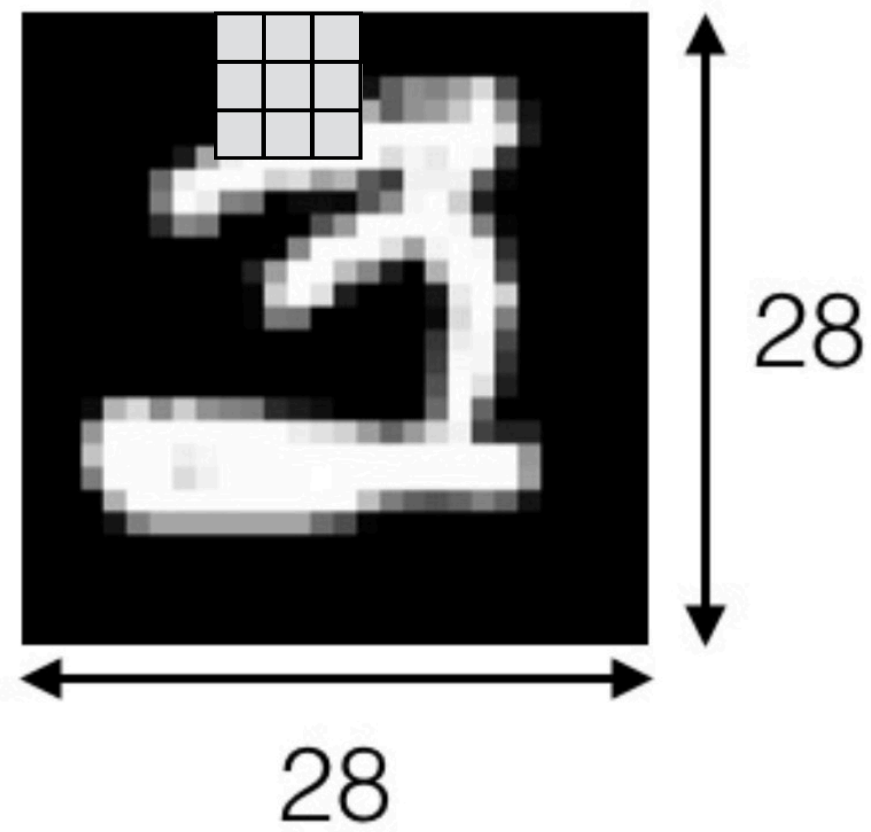


```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

```
, strides=(2, 2),
```

optional parameter
default is (1,1)

with (2,2) stride,
convolution has gaps



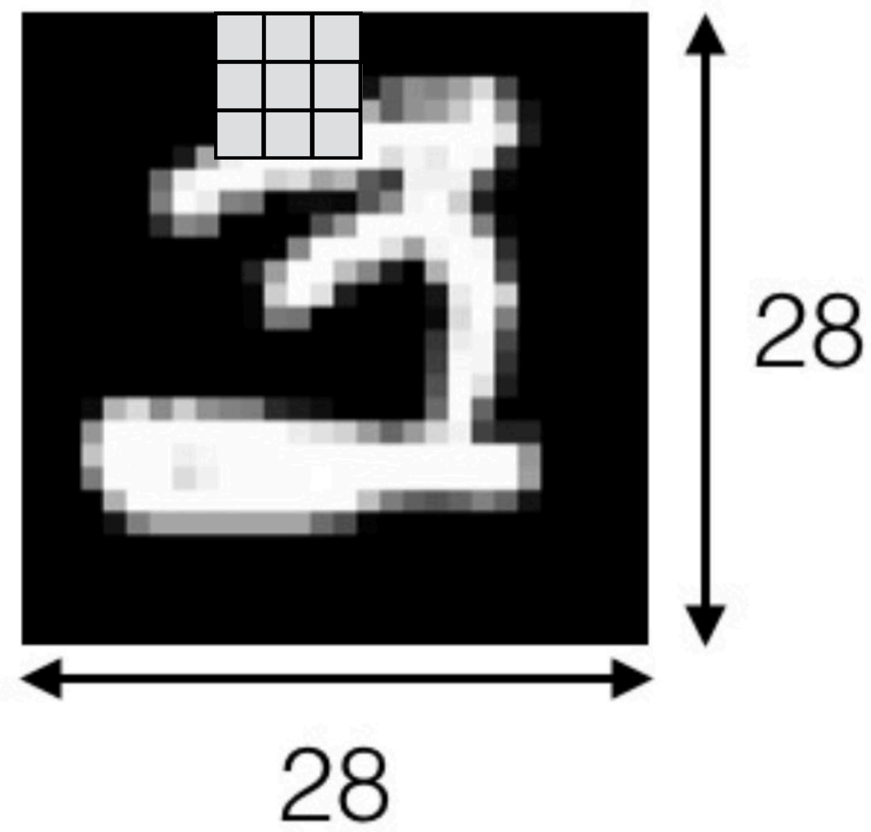
```
# this initializes a blank Sequential network
network = models.Sequential()

# first convolutional layer
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
```

, strides=(2, 2),

optional parameter
default is (1,1)

what will be the dimensions of the Conv2D layer?

with (2,2) stride,
convolution has gaps



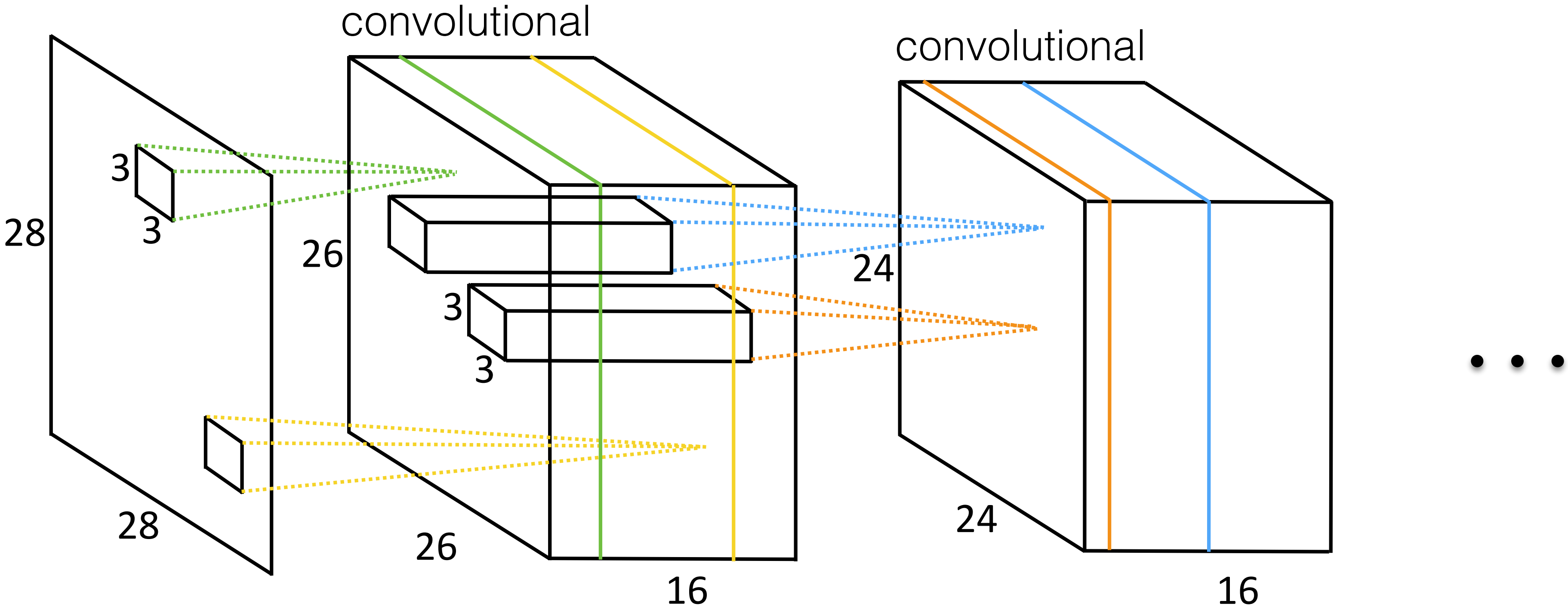
```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), strides=(2,2), activation='relu',
                                   input_shape=(sz,sz,1)))

for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                         format(layer.name,   layer.input.shape, layer.output.shape))
```

```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 13, 13, 16)
```

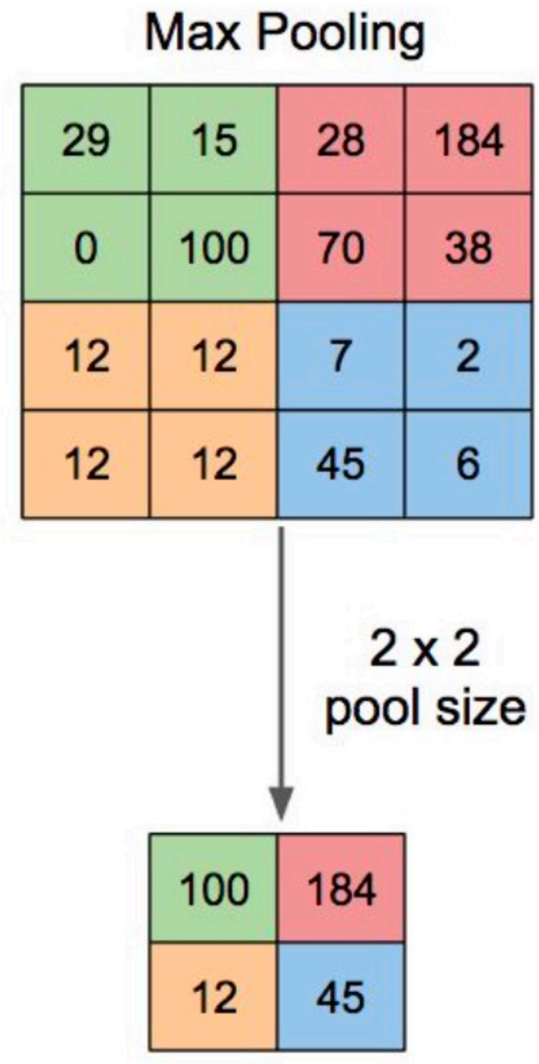
```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
network.add(layers.Conv2D(16, (3,3), activation='relu'))
```

```
for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                        format(layer.name,   layer.input.shape, layer.output.shape))
```

```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 26, 26, 16)
layer name : conv2d_2 | input shape : (None, 26, 26, 16) | output shape : (None, 24, 24, 16)
```


convolutional
convolutional
28
28
3
3
26
26
16
3
3
24
24
16
• • •

```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
network.add(layers.Conv2D(16, (3,3), activation=‘relu'))
network.add(layers.MaxPooling2D((2,2)))
```



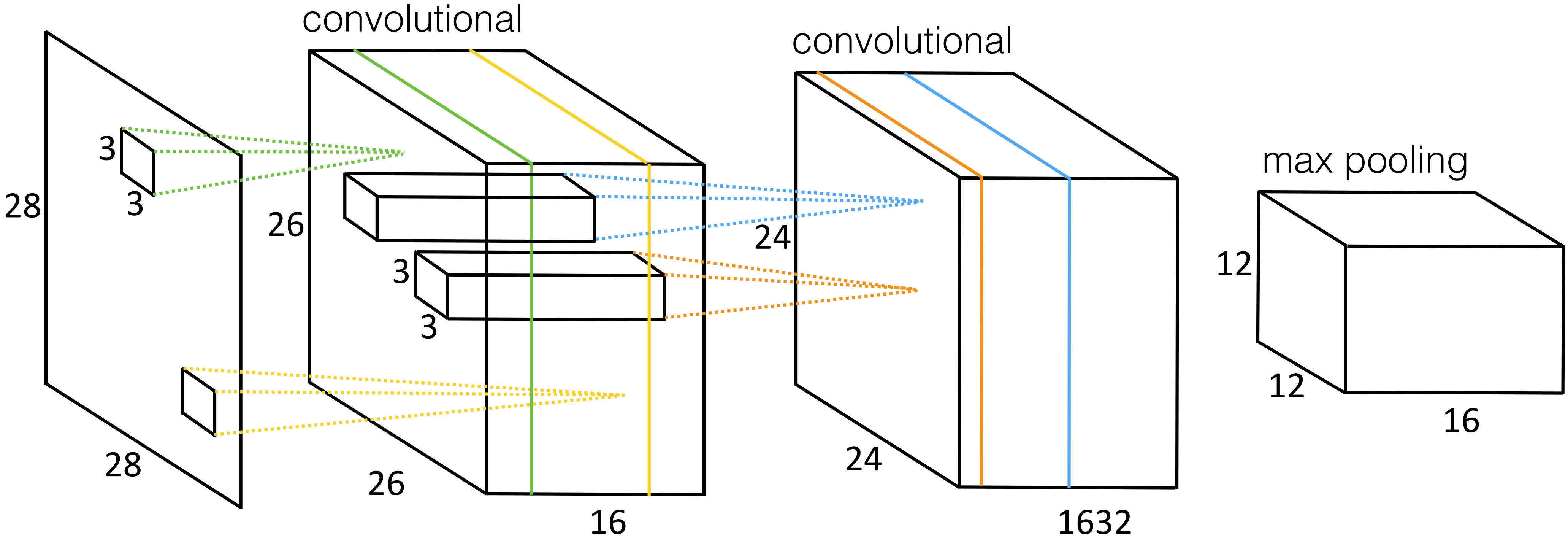
29	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

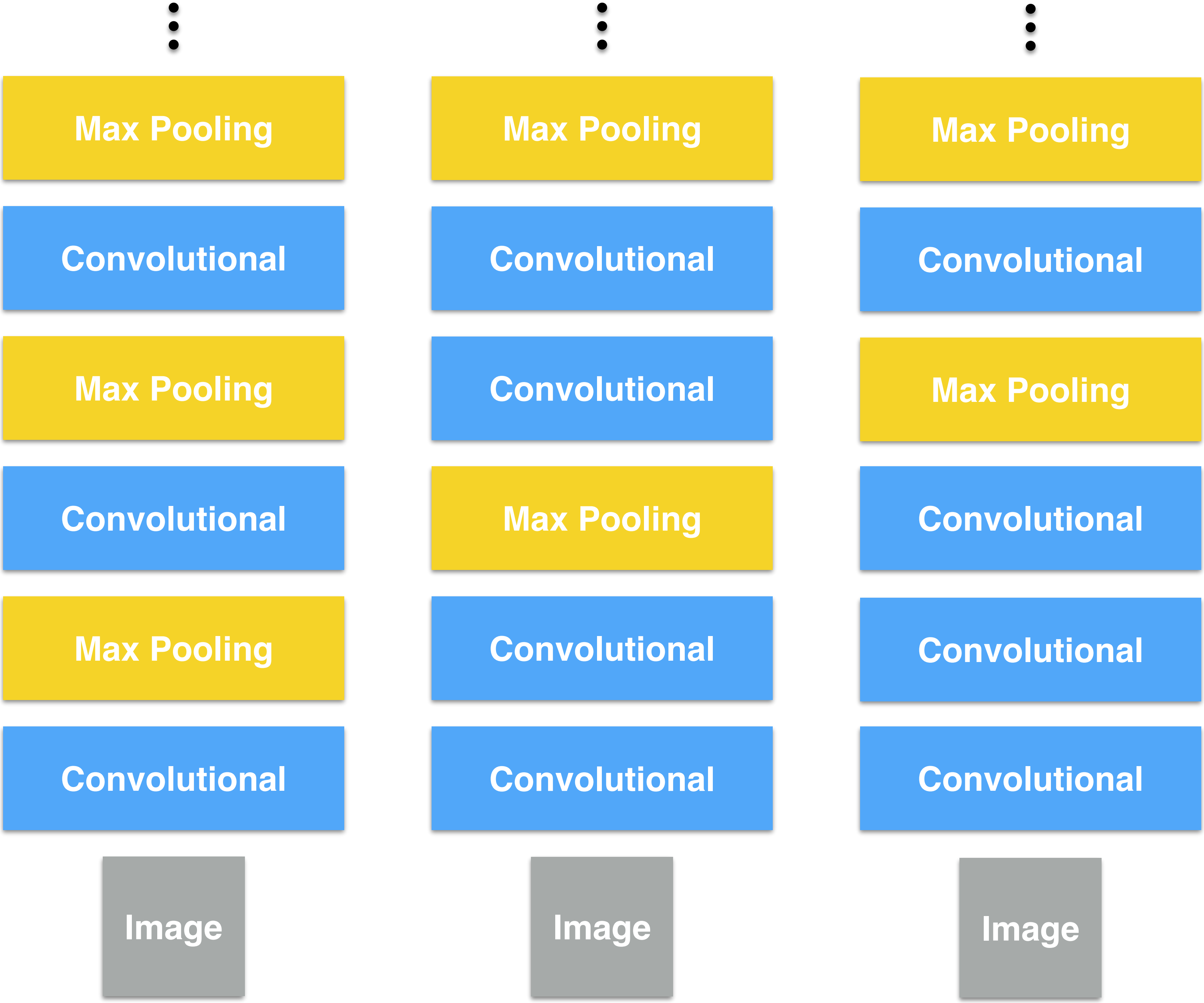
100	184
12	45

what will be the dimensions of the **max_pooling2d** layer?

```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
network.add(layers.Conv2D(16, (3,3), activation=‘relu'))
network.add(layers.MaxPooling2D((2,2)))

for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                        format(layer.name,   layer.input.shape, layer.output.shape))
```

```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 26, 26, 16)
layer name : conv2d_2 | input shape : (None, 26, 26, 16) | output shape : (None, 24, 24, 16)
layer name : max_pooling2d_3 | input shape : (None, 24, 24, 16) | output shape : (None, 12, 12, 16)
```


convolutional
convolutional
max pooling
28
3
3
28
26
3
3
26
16
24
24
1632
12
12
16



⋮

Max Pooling

Convolutional

Max Pooling

Convolutional

Max Pooling

Convolutional

Image

⋮

Max Pooling

Convolutional

Convolutional

Max Pooling

Convolutional

Convolutional

Image

⋮

Max Pooling

Convolutional

Max Pooling

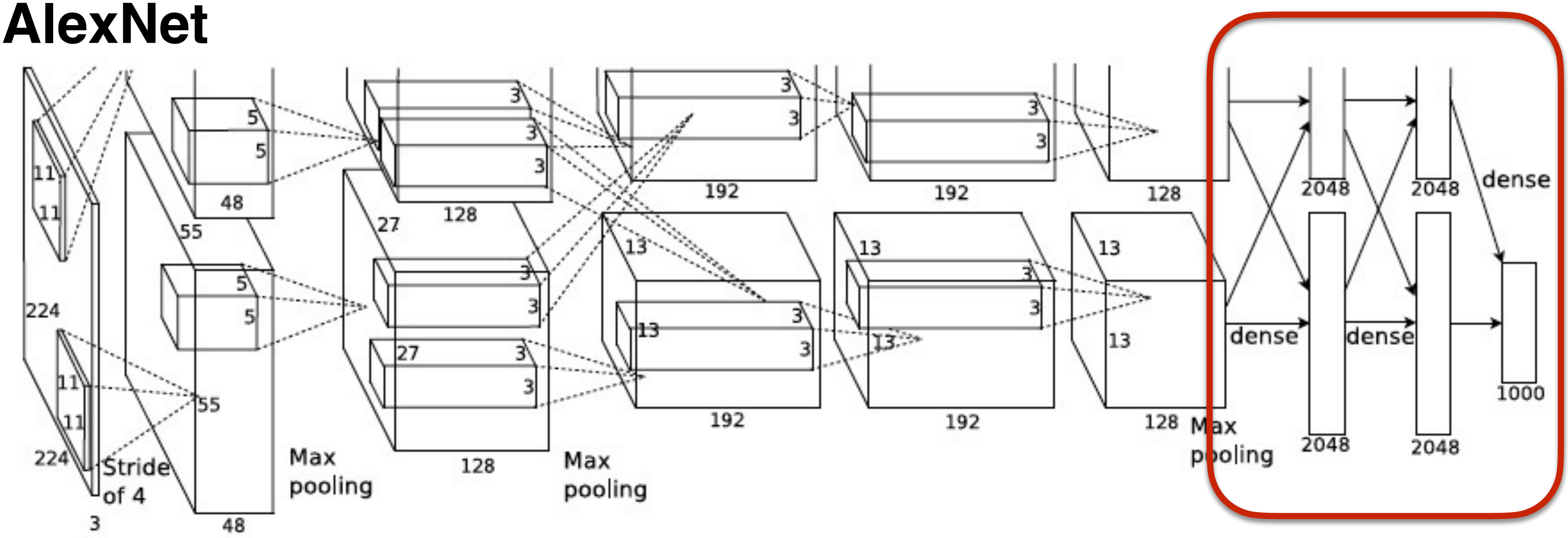
Convolutional

Convolutional

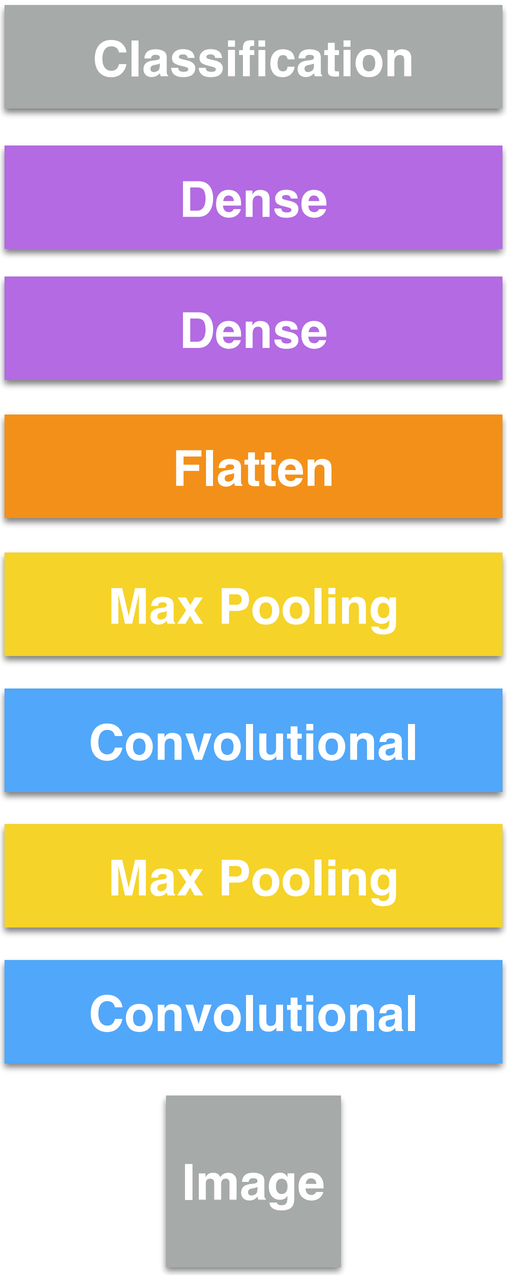
Convolutional

Image

AlexNet


11
11
224
224
11
11
3
Stride of 4
55
55
48
48
5
5
5
5
Max pooling
27
27
128
128
3
3
3
3
3
3
Max pooling
192
192
13
13
3
3
192
192
13
13
3
3
128
128
13
13
3
3
Max pooling
dense
dense
dense
2048
2048
2048
2048
1000

just an example



Classification

Dense

Dense

Flatten

Max Pooling

Convolutional

Max Pooling

Convolutional

Image

```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
network.add(layers.MaxPooling2D((2,2)))
network.add(layers.Conv2D(16, (3,3), activation='relu'))
network.add(layers.MaxPooling2D((2,2)))
network.add(layers.Flatten())
network.add(layers.Dense(20, activation='relu'))
network.add(layers.Dense(nout, activation='softmax'))

for layer in network.layers:
    print('layer name : {} | input shape : {} | output shape : {}'.
                                  format(layer.name,   layer.input.shape, layer.output.shape))
```

```
layer name : conv2d_1 | input shape : (None, 28, 28, 1) | output shape : (None, 26, 26, 16)
layer name : max_pooling2d_2 | input shape : (None, 26, 26, 16) | output shape : (None, 13, 13, 16)
layer name : conv2d_3 | input shape : (None, 13, 13, 16) | output shape : (None, 11, 11, 16)
layer name : max_pooling2d_4 | input shape : (None, 11, 11, 16) | output shape : (None, 5, 5, 16)
layer name : flatten_5 | input shape : (None, 5, 5, 16) | output shape : (None, 400)
layer name : dense_6 | input shape : (None, 400) | output shape : (None, 20)
layer name : dense_7 | input shape : (None, 20) | output shape : (None, 10)
```

```
print(network.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 16)        160
_________________________________________________________________
max_pooling2d_2 (MaxPooling  (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 11, 16)        2320
_________________________________________________________________
max_pooling2d_4 (MaxPooling  (None, 5, 5, 16)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_6 (Dense)              (None, 20)                8020
_________________________________________________________________
dense_7 (Dense)              (None, 10)                210
=================================================================
Total params: 10,710
Trainable params: 10,710
Non-trainable params: 0

___________________________________________________________________________
```

```
network = models.Sequential()
network.add(layers.Conv2D(16, (3,3), activation='relu', input_shape=(sz,sz,1)))
network.add(layers.MaxPooling2D((2,2)))
network.add(layers.Conv2D(16, (3,3), activation='relu'))
network.add(layers.MaxPooling2D((2,2)))
network.add(layers.Flatten())
network.add(layers.Dense(20, activation='relu'))
network.add(layers.Dense(nout, activation='softmax'))

network.compile(optimizer='adam', loss='categorical_crossentropy',
                        metrics=['accuracy'])

history = network.fit(train_images, train_labels_onehot, verbose=True,
                                  validation_split=.1, epochs=ep, batch_size=128)

test_loss, test_accuracy = network.evaluate(test_images, test_labels_onehot)
```