Image Processing with Keras in Python

Saturday, 22 August 2020 9:35 PM

Image Processing With Neural Networks

Images as data

```
import matplotlib.pyplot as plt
data = plt.imread('stop_sign.jpg')
plt.imshow(data)
plt.show()
```

Images as data

```
data.shape
```

```
(2832, 4256, 3)
```

Images as data

```
data[1000, 1500]
```

```
array([0.73333333, 0.07843137, 0.14509804])
```

Images as data

```
data[250, 3500]
```

```
array([0.25882353, 0.43921569, 0.77254902])
```

Modifying image data

```
data[:, :, 1] = 0
data[:, :, 2] = 0
plt.imshow(data)
plt.show()
```

For example, here we set the green and blue values of the pixels to zero.

Changing an image

```
data[200:1200, 200:1200, :] = [0, 1, 0]
plt.imshow(data)
plt.show()
```

Black and white images

```
tshirt[10:20, 15:25] = 1

plt.imshow(tshirt)
plt.show()
```

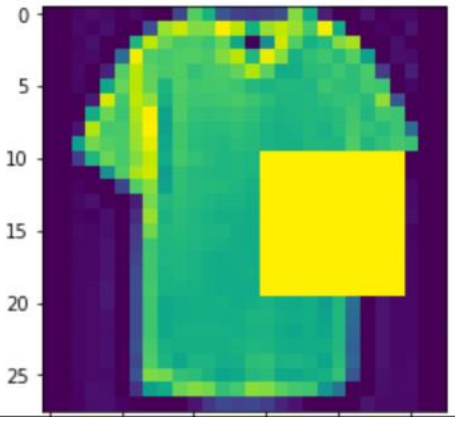


Image classification: training

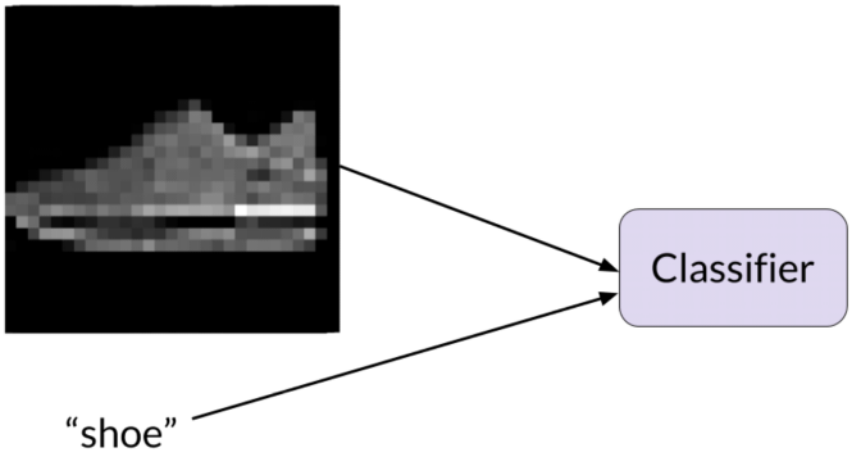
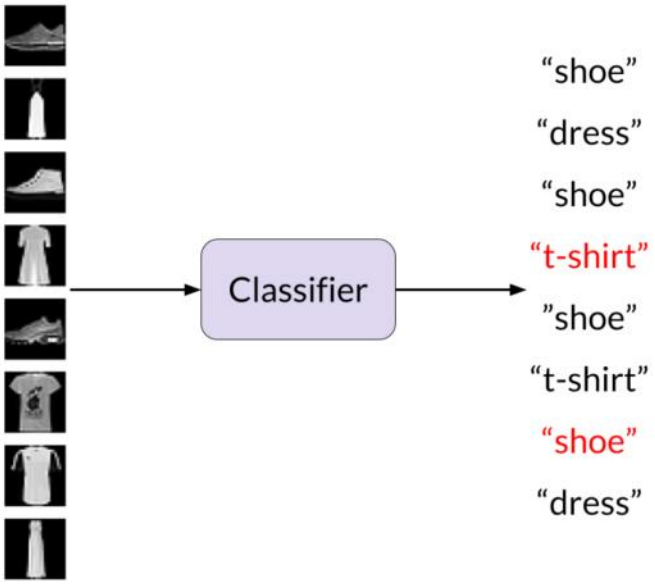


Image classification: evaluation



Representing class data: one-hot encoding

```
labels = ["shoe", "dress", "shoe", "t-shirt",
          "shoe", "t-shirt", "shoe", "dress"]
```

Representing class data: one-hot encoding

```
array([[0., 0., 1.],    <= shoe
       [0., 1., 0.],    <= dress
       [0., 0., 1.],    <= shoe
       [1., 0., 0.],    <= t-shirt
       [0., 0., 1.],    <= shoe
       [1., 0., 0.],    <= t-shirt
       [0., 0., 1.],    <= shoe
       [0., 1., 0.]])   <= dress
```

One-hot encoding

```
categories = np.array(["t-shirt", "dress", "shoe"])
n_categories = 3
ohe_labels = np.zeros((len(labels), n_categories))
for ii in range(len(labels)):
    jj = np.where(categories == labels[ii])
    ohe_labels[ii, jj] = 1
```

One-hot encoding: testing predictions

```
test
```

```
array([[0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.]])
```

```
prediction
```

```
array([[0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.], <= incorrect
       [0., 0., 1.],
       [1., 0., 0.], <= incorrect
       [0., 0., 1.],
       [0., 1., 0.]])
```

```
(test * prediction).sum()
```

```
6.0
```

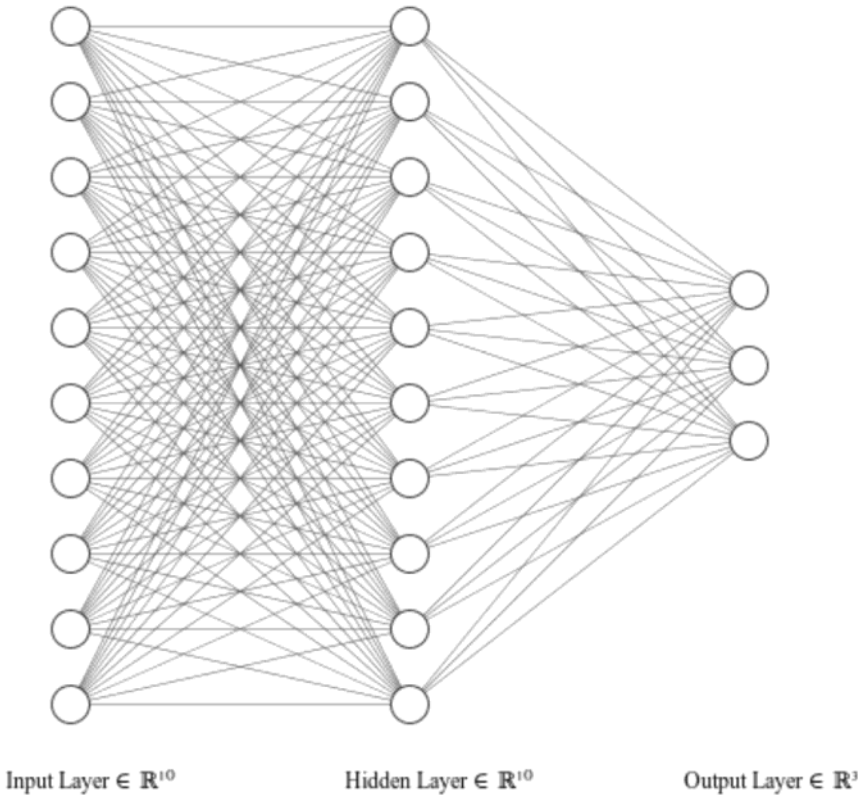
Keras for image classification

```
from keras.layers import Dense
train_data.shape
```

```
(50, 28, 28, 1)
```

Keras for image classification

```
model.add(Dense(10, activation='relu',
          input_shape=(784,)))
model.add(Dense(10, activation='relu'))
model.add(Dense(3, activation='softmax'))
```



Keras for image classification

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

Keras for image classification

```
train_data = train_data.reshape((50, 784))
```

Keras for image classification

```
model.fit(train_data, train_labels,
          validation_split=0.2,
          epochs=3)
```


```
model.fit(train_data, train_labels,
          validation_split=0.2,
          epochs=3)
```


```
Train on 40 samples, validate on 10 samples
Epoch 1/3
32/40 [=========>......] - ETA: 0s - loss: 1.0117 - acc: 0.4688
40/40 [================] - 0s 4ms/step - loss: 1.0438 - acc: 0.4250
                          - val_loss: 0.9668 - val_acc: 0.4000
Epoch 2/3
32/40 [=========>......] - ETA: 0s - loss: 0.9556 - acc: 0.5312
40/40 [================] - 0s 195us/step - loss: 0.9404 - acc: 0.5750
                        - val_loss: 0.9068 - val_acc: 0.4000
Epoch 3/3
32/40 [=========>......] - ETA: 0s - loss: 0.9143 - acc: 0.5938
40/40 [================] - 0s 189us/step - loss: 0.8726 - acc: 0.6750
                        - val_loss: 0.8452 - val_acc: 0.4000
```

Keras for image classification

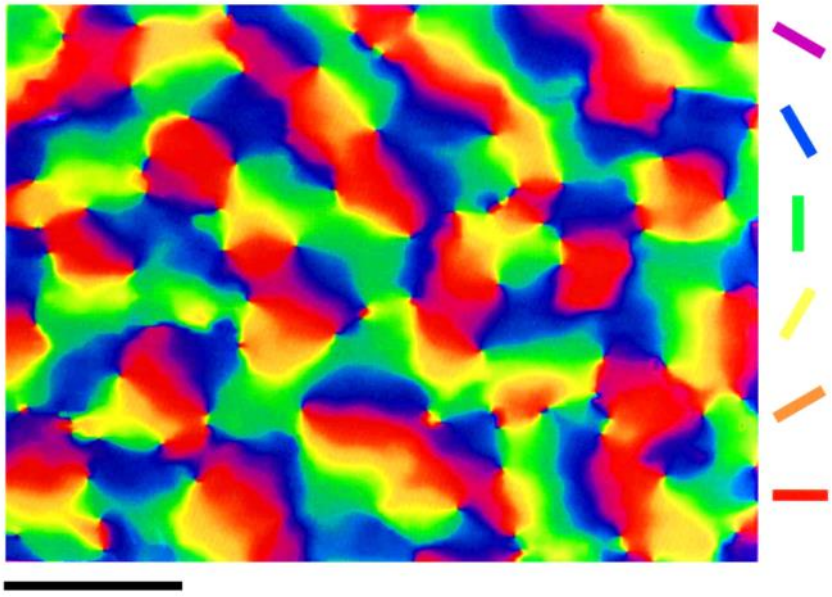
```
test_data = test_data.reshape((10, 784))
model.evaluate(test_data, test_labels)
```

Using Convolutions

Using correlations in images

- Natural images contain spatial correlations
- For example, pixels along a contour or edge
- How can we use these correlations?

Biological inspiration



What is a convolution?

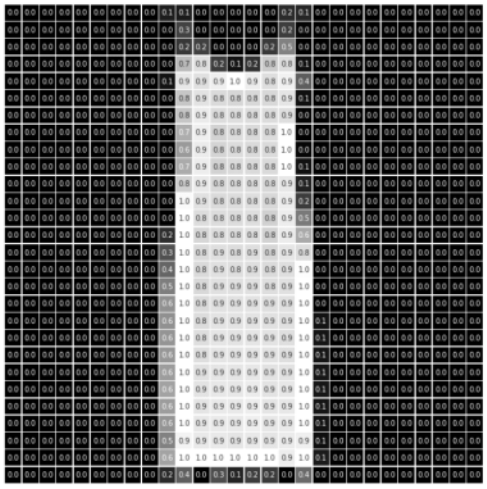
```
array = np.array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
kernel = np.array([-1, 1])
conv = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0])
conv[0] = (kernel * array[0:2]).sum()
conv[1] = (kernel * array[1:3]).sum()
conv[2] = (kernel * array[2:4]).sum()
...
for ii in range(8):
    conv[ii] = (kernel * array[ii:ii+2]).sum()
conv
```

Convolution in one dimension

```
array = np.array([0, 0, 1, 1, 0, 0, 1, 1, 0, 0])
kernel = np.array([-1, 1])
conv = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0])
for ii in range(8):
    conv[ii] = (kernel * array[ii:ii+2]).sum()
conv
```

```
array([ 0,  1,  0, -1,  0,  1,  0, -1,  0])
```

Image convolution



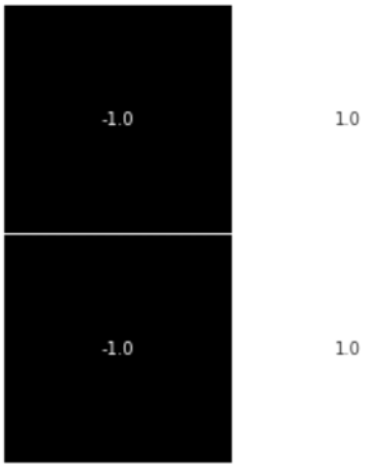
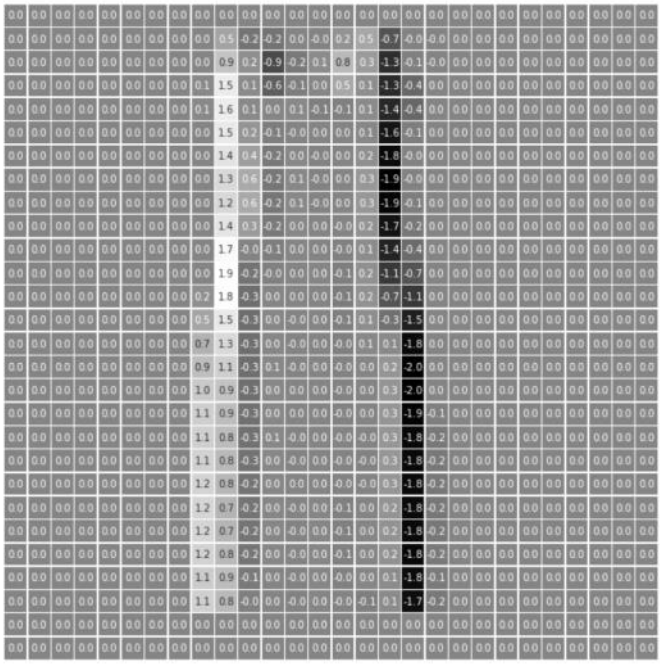


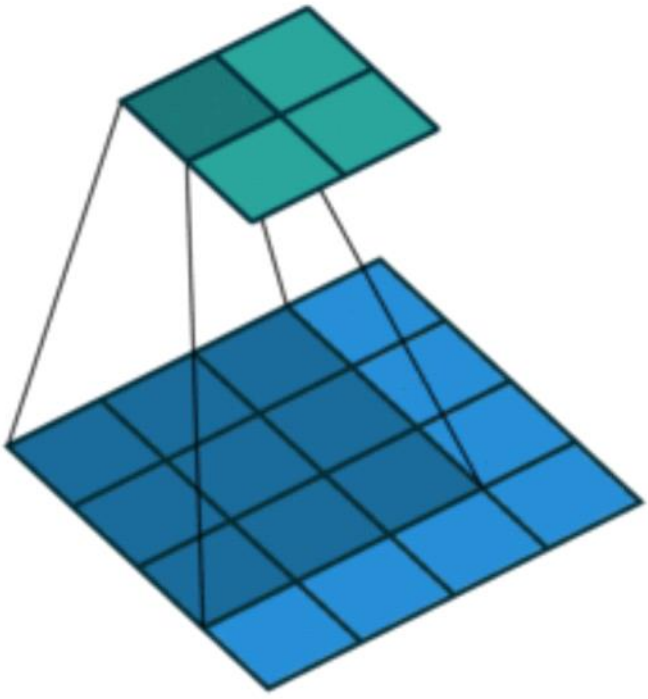
Image convolution



Two-dimensional convolution

```
kernel = np.array([[-1, 1],
                   [-1, 1]])


conv = np.zeros((27, 27)
for ii in range(27):
    for jj in range(27):
        window = image[ii:ii+2, jj:jj+2]
        conv[ii, jj] = np.sum(window * kernel)
```

Convolution



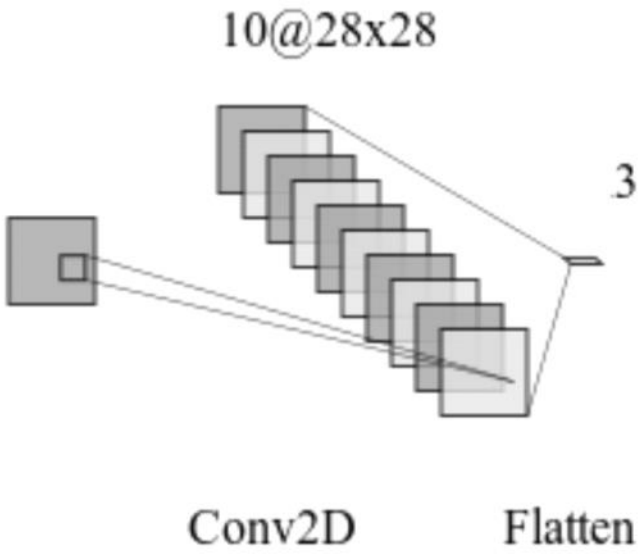
Keras Convolution layer

```
from keras.layers import Conv2D
Conv2D(10, kernel_size=3, activation='relu')
```

Integrating convolution layers into a network

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten

model = Sequential()
model.add(Conv2D(10, kernel_size=3, activation='relu',
          input_shape=(img_rows, img_cols, 1)))

model.add(Flatten())
model.add(Dense(3, activation='softmax'))
```

Our CNN



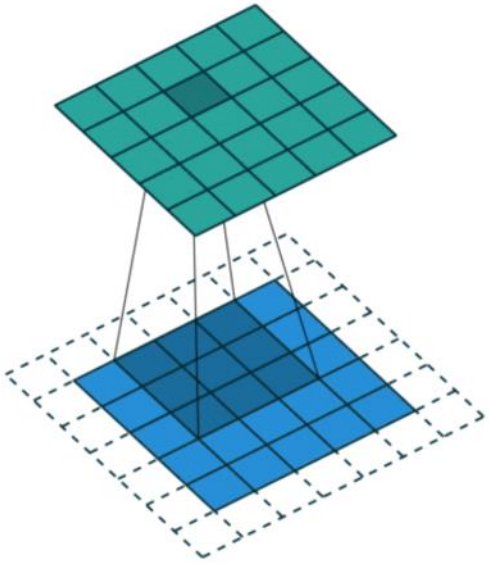
Fitting a CNN

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

train_data.shape
```

```
(50, 28, 28, 1)
```

```
model.fit(train_data, train_labels, validation_split=0.2,
          epochs=3)

model.evaluate(test_data, test_labels, epochs=3)
```

Convolution with zero padding



Zero padding in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
                 padding='valid')
```

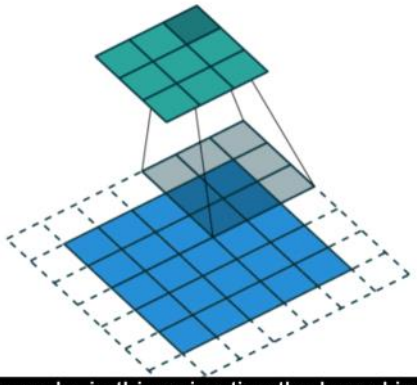
If we provide the value "valid", no zero padding is added.

Zero padding in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
                 padding='same')
```

this layer, so that the output of the convolution has the same size as the input into the convolution.

Strides



For example, in this animation the kernel is strided by two pixels in each step.

Strides in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
          strides=1)
```

The default is for the stride to be set to 1.

Strides in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
          strides=2)
```

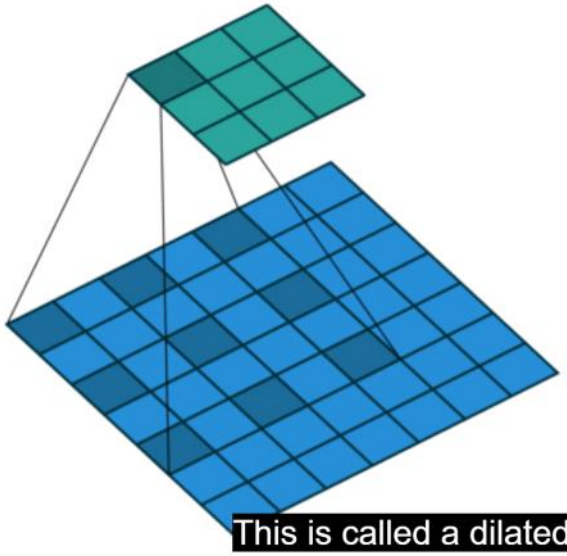
Calculating the size of the output

$$O = ((I - K + 2P)/S) + 1$$

where

- I = size of the input
- K = size of the kernel
- P = size of the zero padding
- S = strides

Dilated convolutions



This is called a dilated convolution.

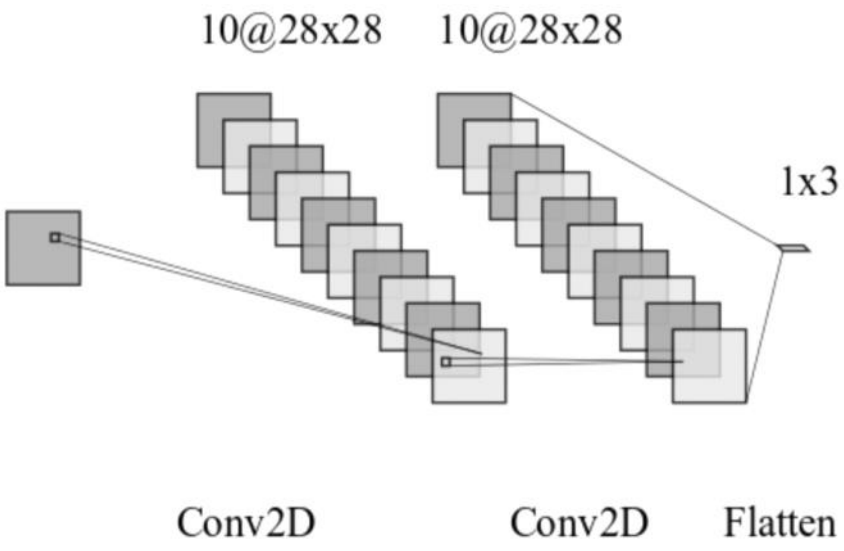
In this case, the convolution kernel has only 9 parameters, but it has the same field of view as a kernel that would have the size 5 by 5.

Dilation in Keras

```
model.add(Conv2D(10, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)),
          dilation_rate=2)
```

Going Deeper

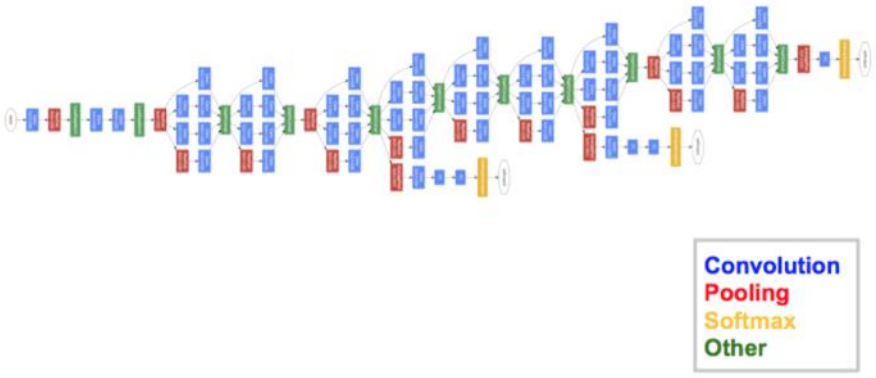
Building a deeper network



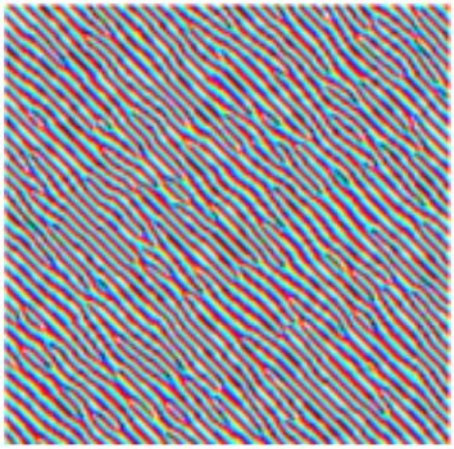
Building a deep network

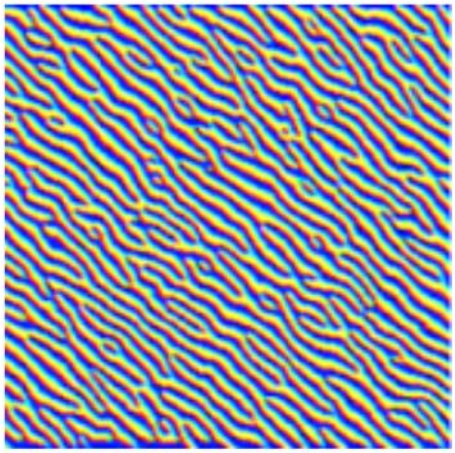
```
model = Sequential()
model.add(Conv2D(10, kernel_size=2, activation='relu',
                 input_shape=(img_rows, img_cols, 1),
                 padding='equal'))
# Second convolutional layer
model.add(Conv2D(10, kernel_size=2, activation='relu')
model.add(Flatten())
model.add(Dense(3, activation='softmax'))
```

Why do we want deep networks?

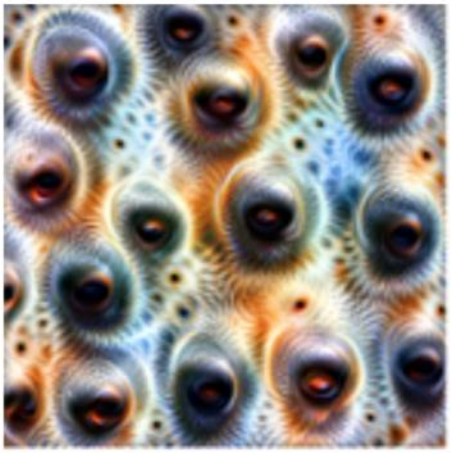


Features in early layers





Features in intermediate layers

Features in late layers


How deep?

- Depth comes at a computational cost
- May require more data

Counting parameters

```
model = Sequential()

model.add(Dense(10, activation='relu',
                input_shape=(784,)))

model.add(Dense(10, activation='relu'))

model.add(Dense(3, activation='softmax'))
```

```
# Call the summary method
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                7850
_________________________________________________________________
dense_2 (Dense)              (None, 10)                110
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 33
=================================================================
Total params: 7,993
Trainable params: 7,993
Non-trainable params: 0
_________________________________________________________________
```

Counting parameters

```
model.add(Dense(
        10, activation='relu',
        input_shape=(784,)))
```

$$parameters = 784 * 10 + 10$$

$$= 7850$$

in this layer, 10 units + 10 parameters for bias terms in every one of these units

```
model.add(Dense(
        10, activation='relu'))
```

$$parameters = 10 * 10 + 10$$

$$= 110$$

In the second layer there is one parameter for every unit in layer one, times the number of units in this layer + 10 parameters for bias terms.

```
model.add(Dense(
        3, activation='softmax'))
```

$$parameters = 10 * 3 + 3$$

$$= 33$$

The last layer has one parameter for every unit in layer 2, times the 3 units in this layer + 3 bias terms.

$$7850 + 110 + 33 = 7993$$

That's why the total number of parameters is 7,993.

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                7850
_________________________________________________________________
dense_2 (Dense)              (None, 10)                110
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 33
=================================================================
Total params: 7,993
Trainable params: 7,993
Non-trainable params: 0
_________________________________________________________________
```

The number of parameters in a CNN

```
model.add(
 Conv2D(10, kernel_size=3,
        activation='relu',
        input_shape=(28, 28, 1),
        padding='same'))
model.add(
 Conv2D(10, kernel_size=3,
        activation='relu',
        padding='same'))


model.add(Flatten())
```

$$parameters = 9 * 10 + 10$$

$$= 100$$

.

$$parameters = 10 * 9 * 10 + 10$$

$$= 910$$

$$parameters = 0$$

$$parameters = 7840 * 3 + 3$$

$$= 23523$$

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 10)        100
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 10)        910
_________________________________________________________________
flatten_3 (Flatten)          (None, 7840)              0
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 23523
=================================================================
Total params: 24,533
Trainable params: 24,533
Non-trainable params: 0
```

Increasing the number of units in each layer

```
model = Sequential()

model.add(Dense(5, activation='relu',
                input_shape=(784,), padding='same'))

model.add(Dense(15, activation='relu', padding='same'))

model.add(Dense(3, activation='softmax'))
```

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 5)                 3925
_________________________________________________________________
dense_2 (Dense)              (None, 15)                90
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 48
=================================================================
Total params: 4,063
Trainable params: 4,063
Non-trainable params: 0
_________________________________________________________________
```

Increasing the number of units in each layer

```
model = Sequential()

model.add(Conv2D(5, kernel_size=3, activation='relu',
                 input_shape=(28, 28, 1),
                 padding="same"))

model.add(Conv2D(15, kernel_size=3, activation='relu',
                 padding="same"))

model.add(Flatten())

model.add(Dense(3, activation='softmax'))
```

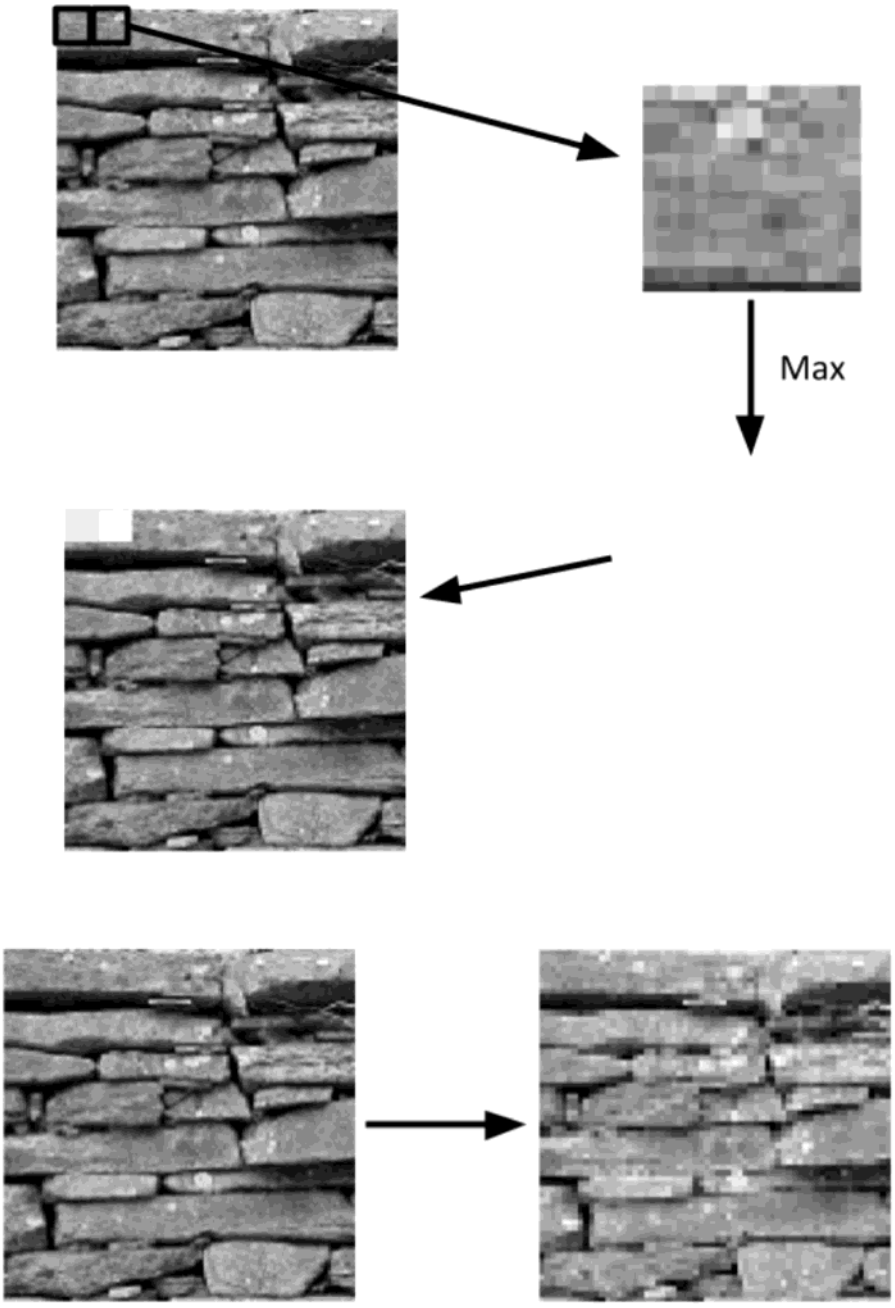
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 28, 28, 5)         50
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 28, 28, 15)        690
_________________________________________________________________
flatten_6 (Flatten)          (None, 11760)             0
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 35283
=================================================================
Total params: 36,023
Trainable params: 36,023
Non-trainable params: 0
```

Pooling operations

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 28, 28, 5)         50
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 28, 28, 15)        690
_________________________________________________________________
flatten_6 (Flatten)          (None, 11760)             0
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 35283
=================================================================
Total params: 36,023
Trainable params: 36,023
Non-trainable params: 0
```


Max

Implementing max pooling

```
result = np.zeros((im.shape[0]//2, im.shape[1]//2))
result[0, 0] = np.max(im[0:2, 0:2])
result[0, 1] = np.max(im[0:2, 2:4])
result[0, 2] = np.max(im[0:2, 4:6])
```

...

```
result[1, 0] = np.max(im[2:4, 0:2])
result[1, 1] = np.max(im[2:4, 2:4])
```

Implementing max pooling

```
for ii in range(result.shape[0]):
    for jj in range(result.shape[1]):
        result[ii, jj] = np.max(im[ii*2:ii*2+2, jj*2:jj*2+2])
```

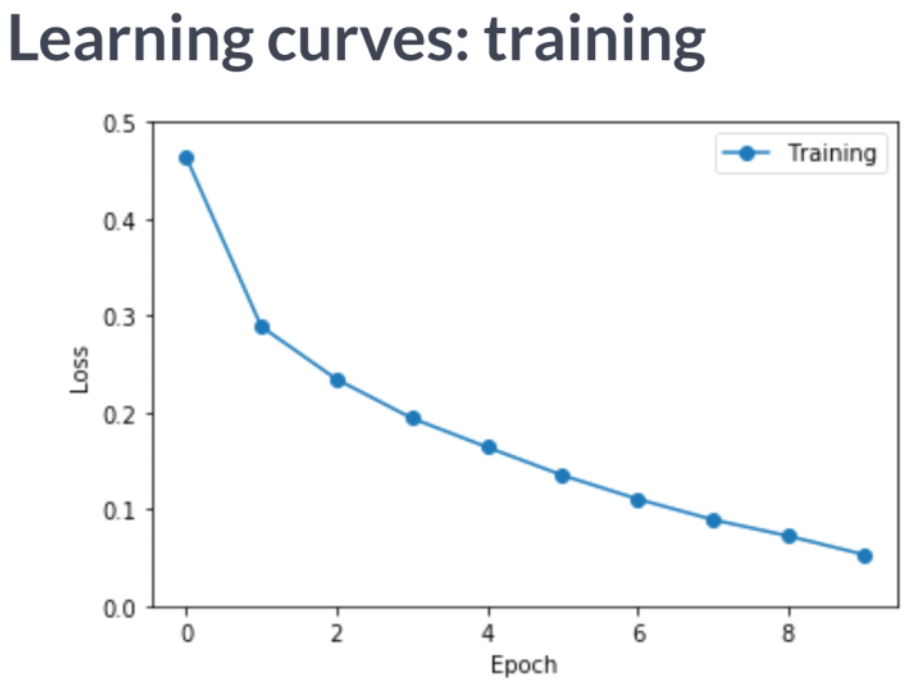
Max pooling in Keras

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPool2D
model = Sequential()
model.add(Conv2D(5, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)))
model.add(MaxPool2D(2))
model.add(Conv2D(15, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)))
model.add(MaxPool2D(2))
model.add(Flatten())
model.add(Dense(3, activation='softmax'))
```

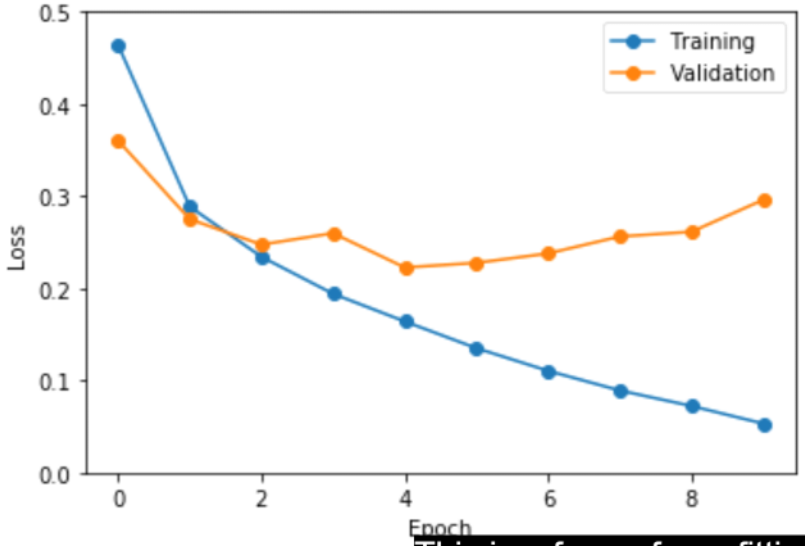
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 5)         50
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 5)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 15)        690
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 15)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 375)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 1128
=================================================================
```

Understanding and Improving Deep Convolutional Networks

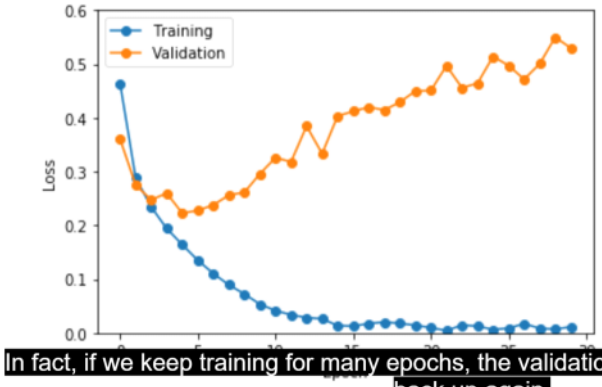


Learning curves: validation



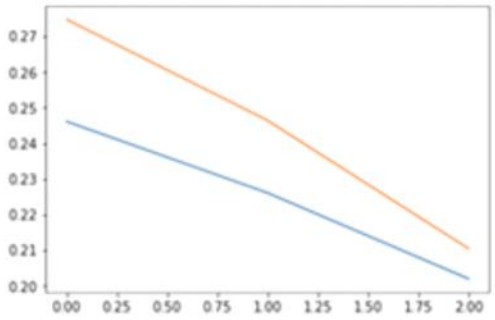
This is a form of overfitting.

Learning curves: overfitting



In fact, if we keep training for many epochs, the validation accuracy can start increasing back up again.

```
training = model.fit(train_data, train_labels,
                     epochs=3, validation_split=0.2)
import matplotlib.pyplot as plt
plt.plot(training.history['loss'])
plt.plot(training.history['val_loss'])
plt.show()
```



Storing the optimal parameters

```
from keras.callbacks import ModelCheckpoint

# This checkpoint object will store the model parameters
# in the file "weights.hdf5"
checkpoint = ModelCheckpoint('weights.hdf5', monitor='val_loss',
                             save_best_only=True)
# Store in a list to be used during training
callbacks_list = [checkpoint]
# Fit the model on a training set, using the checkpoint as a
#callback
model.fit(train_data, train_labels, validation_split=0.2,
          epochs=3, callbacks=callbacks_list)
```

Loading stored parameters

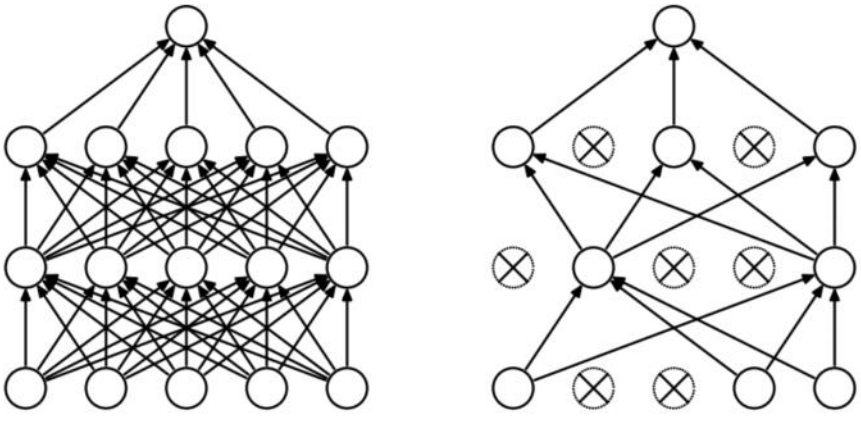
```
model.load_weights('weights.hdf5')
model.predict_classes(test_data)
array([2, 2, 1, 2, 0, 1, 0, 1, 2, 0])
```

Dropout

In each learning step:

- Select a subset of the units
- Ignore it in the forward pass
- And in the back-propagation of error

Dropout



(a) Standard Neural Net

(b) After applying dropout.

Dropout in Keras

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, Dropout

model = Sequential()
model.add(Conv2D(5, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)))

model.add(Dropout(0.25))

model.add(Conv2D(15, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(3, activation='softmax'))
```

Batch normalization

- Rescale the outputs

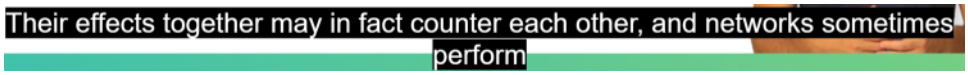
Batch Normalization in Keras

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, BatchNo


model = Sequential()
model.add(Conv2D(5, kernel_size=3, activation='relu',
                 input_shape=(img_rows, img_cols, 1)))

model.add(BatchNormalization())

model.add(Conv2D(15, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(3, activation='softmax'))
```

Be careful when using them together!

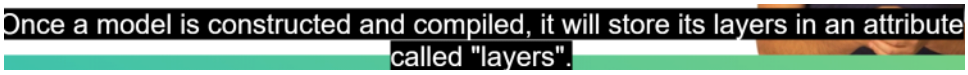
The disharmony between dropout and batch normalization

This is because while dropout slows down learning, making it more



Selecting layers

```
model.layers
```




```
[<keras.layers.convolutional.Conv2D at 0x109f10c18>,
 <keras.layers.convolutional.Conv2D at 0x109ec5ba8>,
 <keras.layers.core.Flatten at 0x1221ffcc0>,
 <keras.layers.core.Dense at 0x1221ffef0>]
```

Getting model weights

```
conv1 = model.layers[0]
weights1 = conv1.get_weights()
len(weights1)
```

```
2
```

```
kernels1 = weights1[0]
kernels1.shape
```

```
(3, 3, 1, 5)
```

The first 2 dimensions denote the kernel size.

The third dimension denotes the number of channels in the kernels.

The last dimension denotes the number of kernels in this layer: 5.

```
kernel1_1 = kernels1[:, :,
                     0, 0]

kernel1_1.shape
```

```
(3, 3)
```

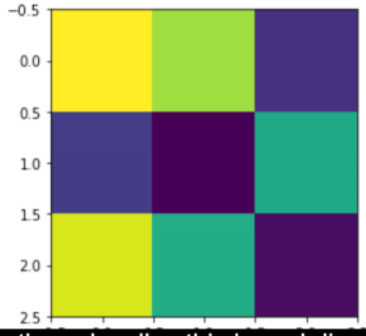
To pull out the first kernel in this layer, we would use the index 0 into the last dimension

Because there is only one channel, we can also index on the channel dimension, to collapse that dimension.

This would return the 3 by 3 array containing this convolutional kernel.

Visualizing the kernel

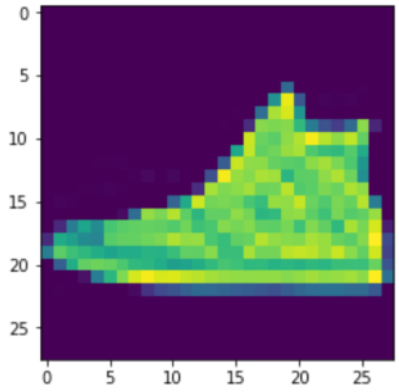
```
plt.imshow(kernel1_1)
```



We can then visualize this kernel directly, but understanding what kinds of features this kernel is responding to may be hard just from direct observation.

Visualizing the kernel responses

```
test_image = test_data[3, :, :, 0]

plt.imshow(test_image)
```

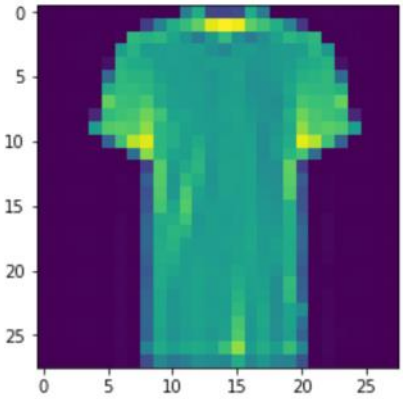


Visualizing the kernel responses

```
filtered_image = convolution(test_image, kernel1_1)

plt.imshow(filtered_image)
```

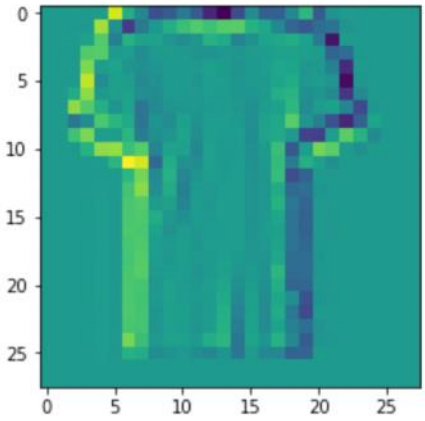
Visualizing the kernel responses

```
test_image = test_data[4, :, :, 1]
plt.imshow(test_image)
```



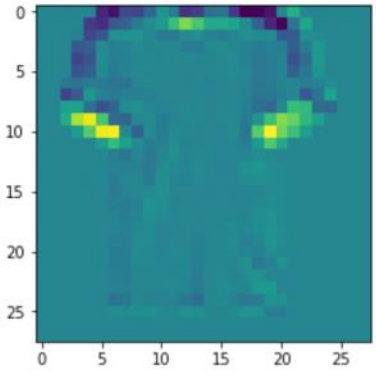
Visualizing the kernel responses

```
filtered_image = convolution(test_image, kernel1_1)
plt.imshow(filtered_img)
```



Visualizing the kernel responses

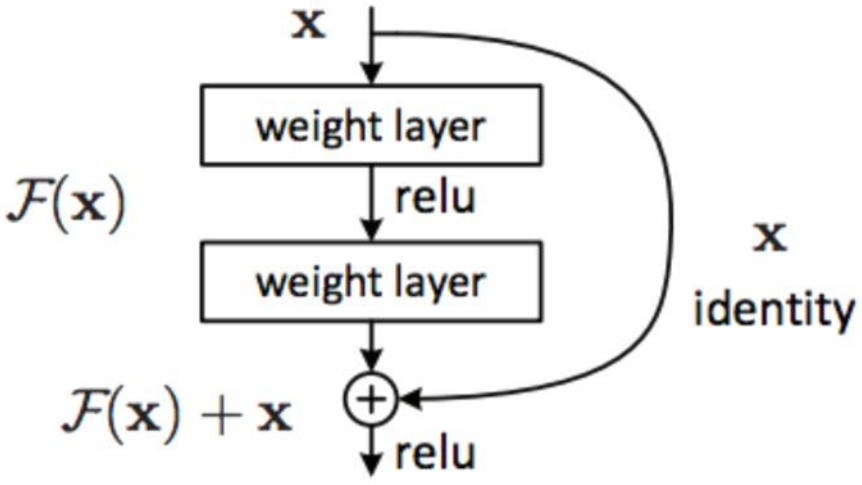
```
kernel1_2 = kernels[:, :, 0, 1]
filtered_image = convolution(test_image, kernel1_2)
plt.imshow(filtered_img)
```



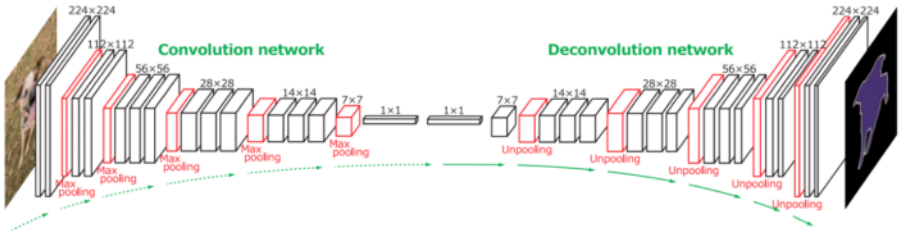
What next?

- Even deeper networks
- Residual networks
- Transfer learning
- Fully convolutional networks

Residual networks



Fully convolutional networks



Generative adversarial networks