CS5489- Machine Learning

Lecture 9a - Convolutional Neural Networks (CNNs)

Dr. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

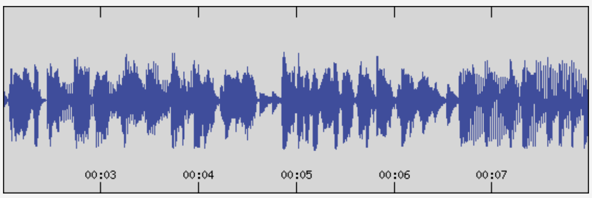
- Convolutional neural network (CNN)
- Regularization

Signals

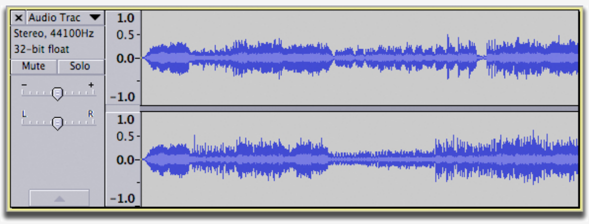
- So far we have assumed the input $\mathbf{x}$ is a vector
  - or have turned 2D images into vectors.
- *What if the input has more structure?*

- For example:
  - **1-D signal** (time)

mono audio (1 feature)



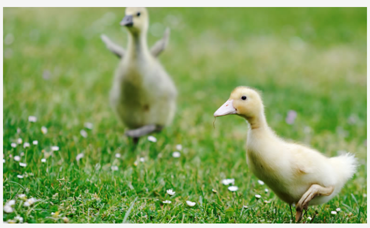
stereo audio (2 features)



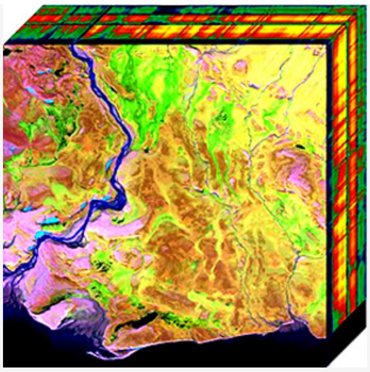
- **2-D signal** (space)

grayscale image (1 feature)

color image (3 features)

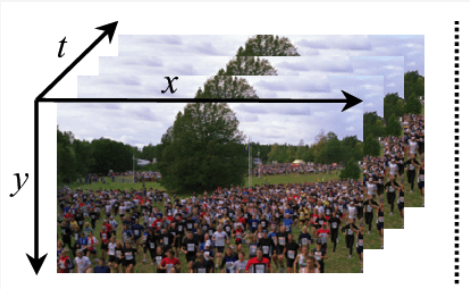


hyperspectral image (300 features)

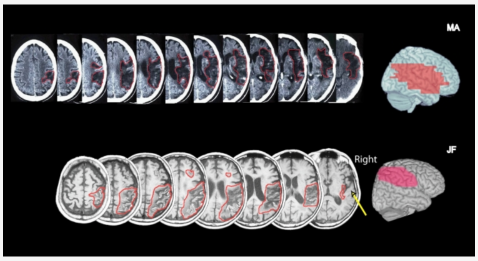


- **3-D signal** (space+time, volume)

color video (3 features)

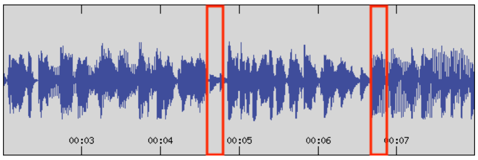


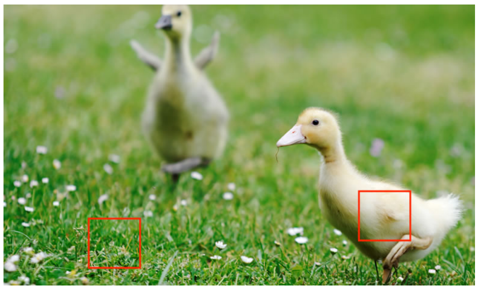
3D CT scan (1 feature)



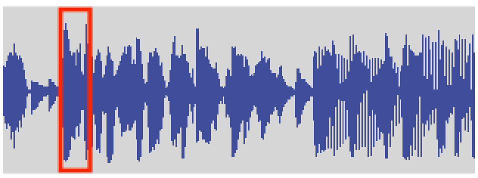
Assumed Properties of Signals

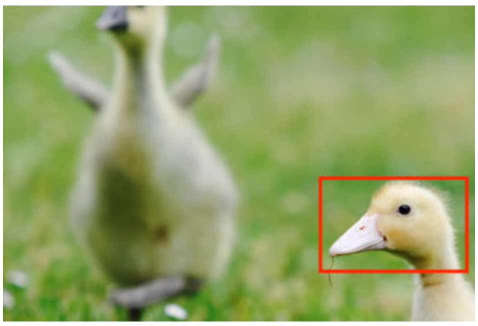
- *Locality*
  - at low-level, features from 1 region are independent (do not depend on) features from a far-away region.

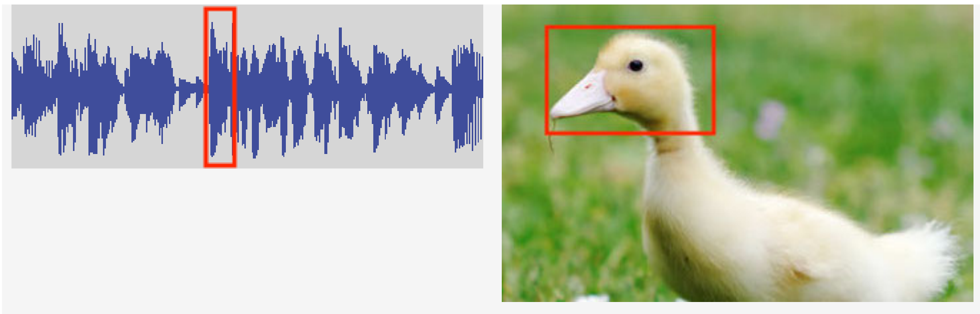




- *Translation*
  - the same features can appear anywhere in the signal.

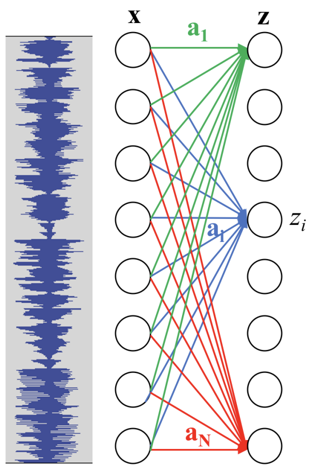




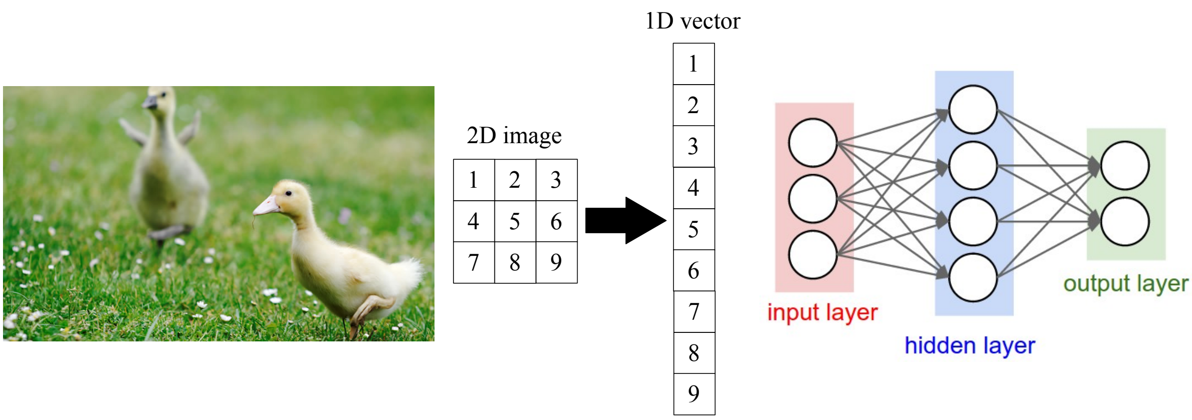


Using the standard MLP layer...

- Each feature z_i is computed from all the inputs, but we only want local features (locality).
- The pattern could appear anywhere, but weights are trained for each location z_i separately (translation).



- For image input, we transform the image into a vector, which is the input into the MLP.



- **Problem:** This ignores the spatial relationship between pixels in the image.
  - Images contain local structures
    - groups of neighboring pixels correspond to visual structures (edges, corners, texture).
    - pixels far from each other are typically not correlated.

- **How to model the local structure of the signal?**
  - local features, feature translation

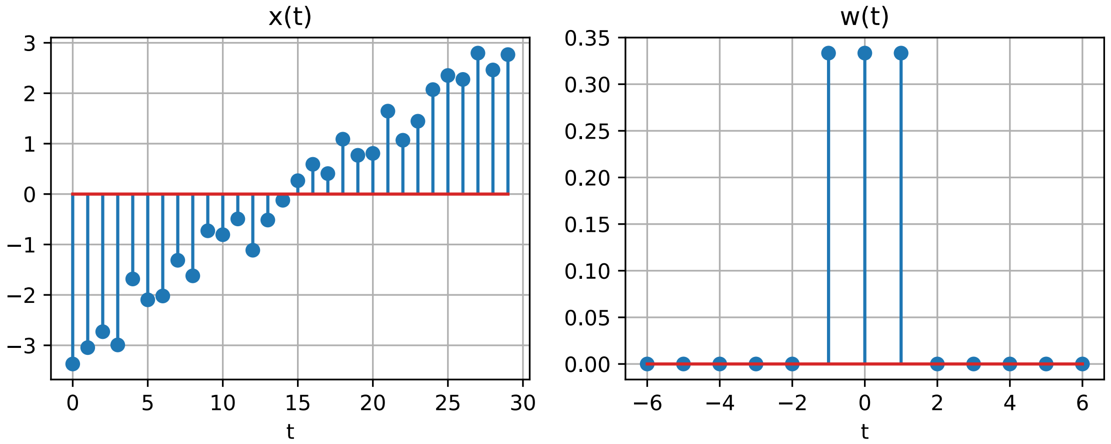
- **Answer**: Use a local feature extractor in the signal.

Convolution

- Consider 1D signal in *discrete* time: $x(t)$, $t \in \mathbb{Z}$
- Define the filter $w(t)$
  - "flipped" filter: $\tilde{w}(t) = w(-t)$

In [9]:
```
xfig
```

Out[9]:

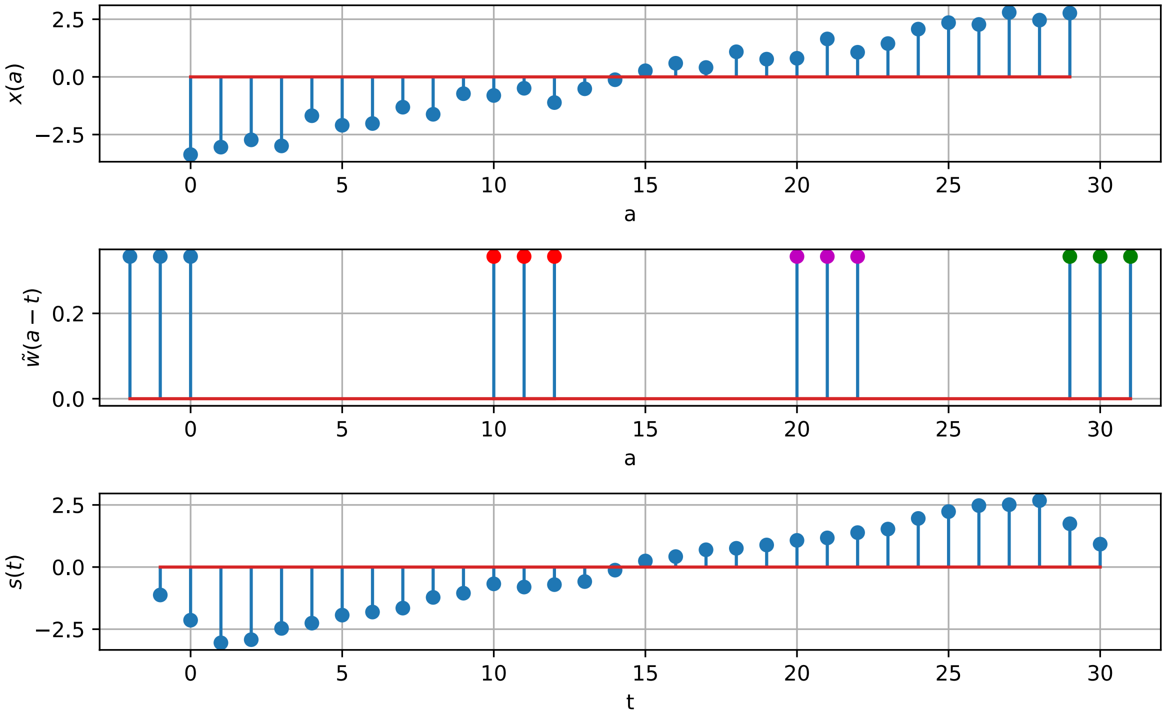


- Convolution is a filtering operation
  - $s(t) = x * w = \sum_a x(a)w(t-a)$
    - "$*$" is the symbol for convolution
- It's related to cross-correlation with the "flipped" filter.
  - $s(t) = \sum_a x(a)\tilde{w}(a-t)$
  - for a given t:
    1. shift $\tilde{w}$ by t
    2. multiply shifted $\tilde{w}$ with x
    3. sum to get $s(t)$

- $s(t) = \sum_a x(a)\tilde{w}(a-t)$
  1. shift $\tilde{w}$ by t
  2. multiply shifted $\tilde{w}$ with x
  3. sum to get $s(t)$

In [11]:
```
sfig
```

Out[11]:

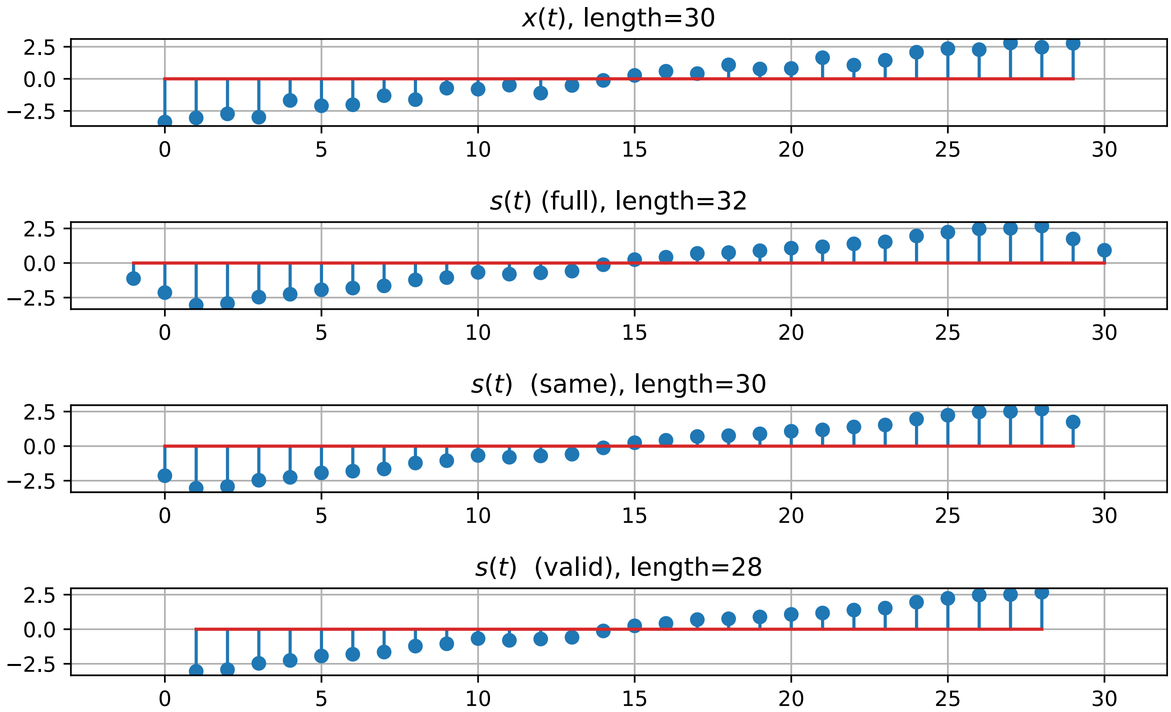


- Boundary conditions
  - It is assumed that the rest of the signal is all 0.
- The convolution result near the ends of the signal uses these "artificial" zeros.
  - the filtered signal is longer than the original signal
  - length increases by $P-1$, where P is the non-zero extent of the filter.

- Three ways to handle this:
  - 1) use everything with *non-zero* response ('full' mode)
  - 2) keep the response the *same* length as the signal ('same' mode)
  - 3) keep only responses where the *entire* filter is on the signal ('valid' mode)

In [13]:
```
cfig
```

Out[13]:



2D Convolution

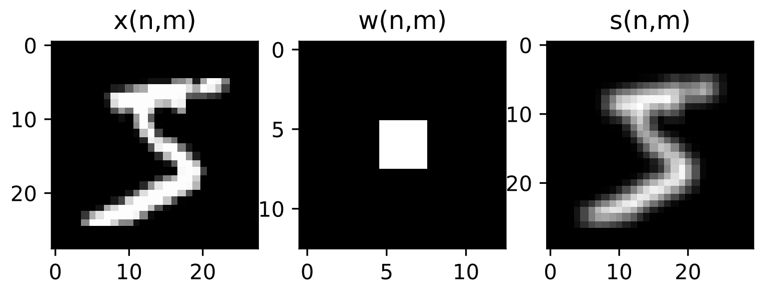
- Straightforward to extend to multiple dimensions
- 2D discrete convolution

$$s(n, m) = x * w = \sum_a \sum_b x(a, b)w(n - a, m - b)$$

In [15]:

```
sfig
```

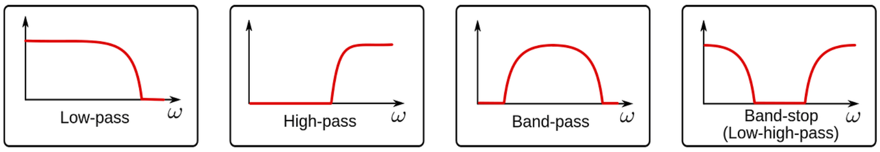
Out[15]:



Two Interpretations of Convolution

- Interpretation 1:
  - w is a filter on the frequency spectrum of signal x.

- Example filters: *low-pass*, *high-pass*, *band-pass*, *moving-average*

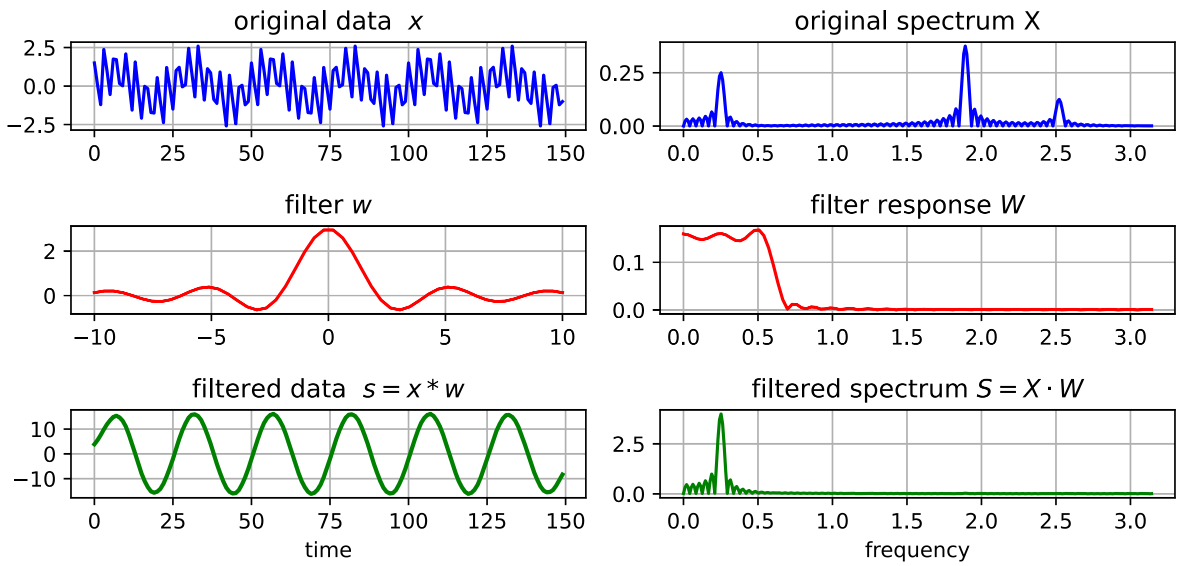


- Analysis is in the frequency domain:
  - **Time domain** $x(t) \Longleftrightarrow$ **Frequency domain** $X(\omega)$
- Discrete-time Fourier transform (DTFT)
  - represent data as sum of complex exponentials basis functions with different frequencies.
    - $e^{-i\omega t} = \cos(\omega t) - i\sin(\omega t)$
    - Imaginary number: $i^2 = -1$
  - $X(\omega) = \sum_t x(t)e^{-i\omega t} \Longleftrightarrow x(t) = \frac{1}{2\pi}\int X(\omega)e^{i\omega t}d\omega$

- **Key results:**
  - convolution in the time domain is equivalent to multiplication in the frequency domain:
    - $s(t) = x(t) * w(t) \Longleftrightarrow S(\omega) = X(\omega)W(\omega)$
  - we can design and analyze filters in the frequency domain, and then obtain their time-domain representation.

In [17]:
```
ffig
```

Out[17]:



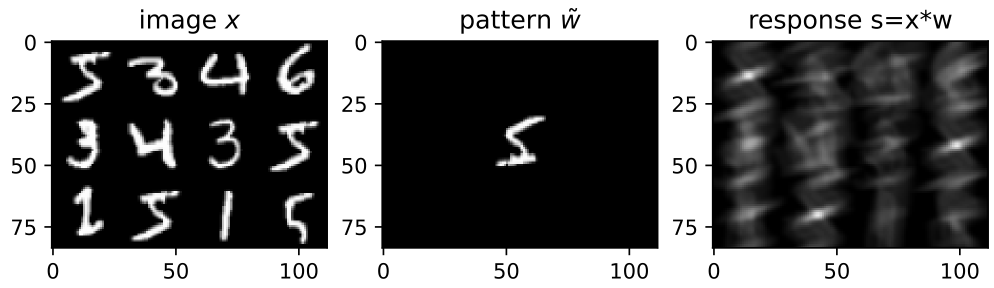
- Relationships between time and frequency spectrum
  - short-duration signal $\Leftrightarrow$ large frequency spectrum
  - long-duration signal $\Leftrightarrow$ small frequency spectrum
- For filters...
  - short (small) filter only captures short-range correlations (high frequencies).

- long (large) filter can capture long-range correlations (low frequencies).

- Interpretation 2:
  - $\tilde{w}$ is a pattern (template); try to find this pattern.
  - the maximum correlation occurs when pattern $\tilde{w}$ matches the local x.
    - for a fixed energy $||x||^2 = 1$, $x = \frac{\tilde{w}}{||\tilde{w}||}$ has the maximum correlation with (response to) $\tilde{w}$.

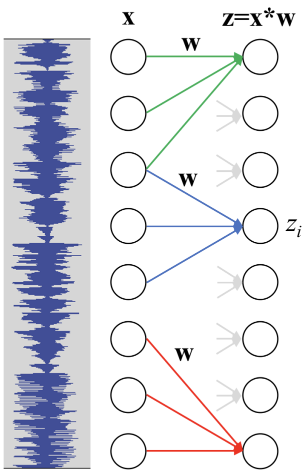
In [20]:
```
pfig
```

Out[20]:



Convolution as a layer

- output layer is the convolution of input with filter (kernel) $\mathbf{w}$.
  - $\mathbf{z} = \mathbf{x} * \mathbf{w}$.
- filter $\mathbf{w}$ acts locally on input $\mathbf{x}$.
  - $\mathbf{w}$ also called a **kernel**.



- Equivalent to a linear transformation where A has a particular form.
  - For example, if $\mathbf{w} = [w_1, w_2, w_3]$ and using "same" mode,

$$\mathbf{z} = \mathbf{x} * \mathbf{w}$$

$$= \mathbf{A}^T\mathbf{x} = \begin{bmatrix} w_2 & w_3 & 0 & 0 & 0 & 0 & \cdots \\ w_1 & w_2 & w_3 & 0 & 0 & 0 & \cdots \\ 0 & w_1 & w_2 & w_3 & 0 & 0 & \cdots \\ 0 & 0 & w_1 & w_2 & w_3 & 0 & \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}$$

- Note: $\mathbf{A}$ has size $|\mathbf{x}||\mathbf{z}|$, but only $|\mathbf{w}|$ parameters.

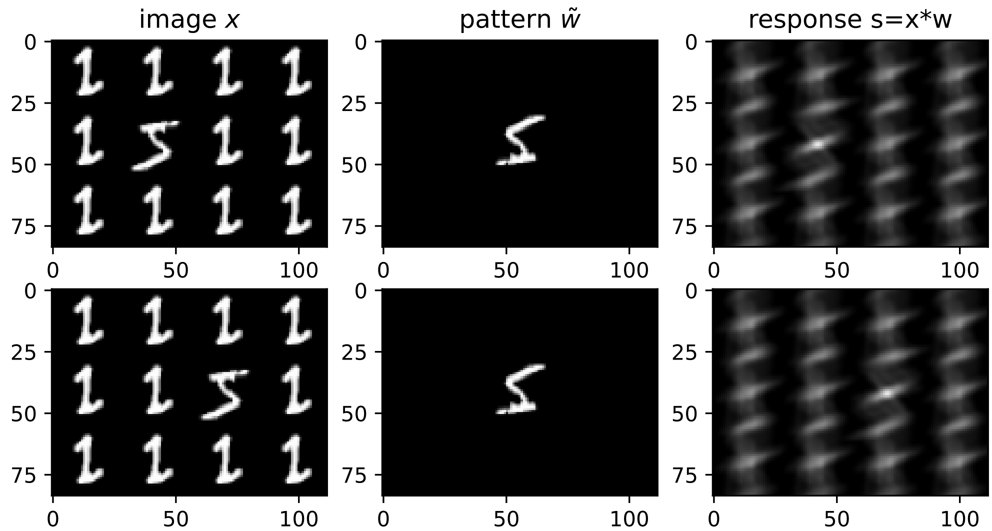
Translation equivariance

- Shifting x also shifts the response s.
- if $s = x * w$,
  - then $s(t-a) = x(t-a) * w(t)$
- We can find the pattern everywhere in x using the same filter.

In [22]:

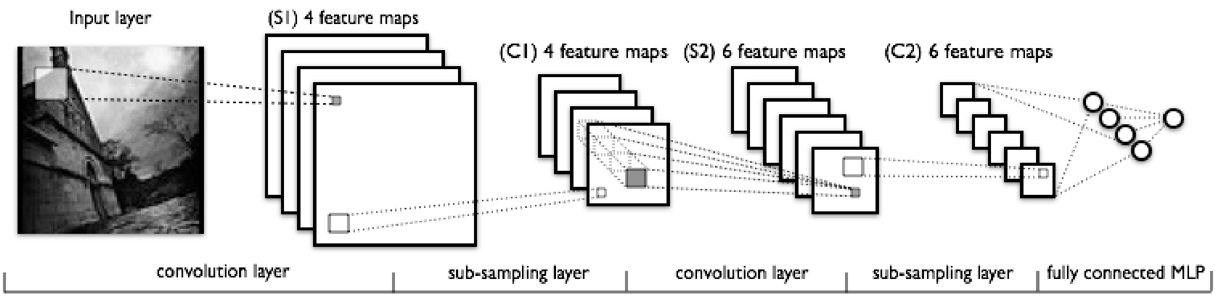
```
cfig
```

Out[22]:



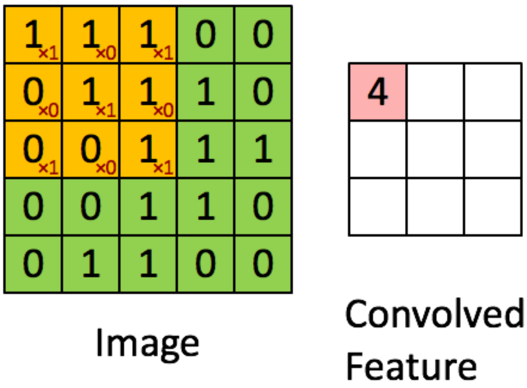
Convolutional Neural Network (CNN)

- series of convolutional layers, sub-sampling layers, and MLP classifier.
  - convolutional and subsampling layers extract image features.
  - MLP uses extracted features for classification.

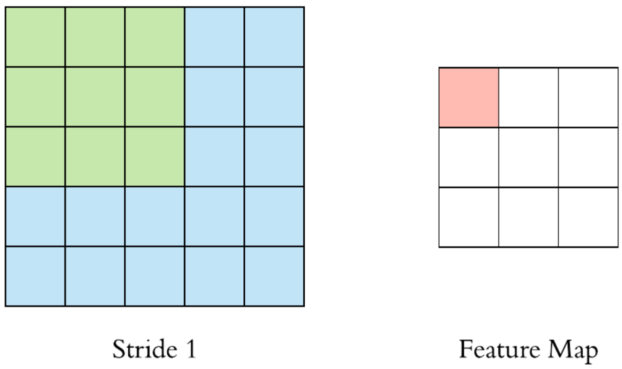


2D Convolution

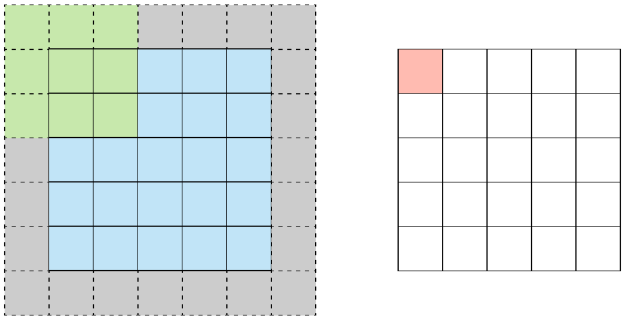
- Use the spatial structure of the image
- 2D convolution filter
  - the weights $\mathbf{W}$ form a 2D filter template
  - filter response: $h = f(\sum_{x,y} W_{x,y} P_{x,y})$
    - $\mathbf{P}$ is an image patch with the same size as $\mathbf{W}$.
- Convolution feature map
  - pass a sliding window over the image, and apply filter to get a *feature map*.



- Convolution modes
  - **"valid"** mode - only compute feature where convolution filter has valid image values.
    - size of feature map is reduced.



- Convolution modes
  - **"same"** mode - zero-pad the border of the image
    - feature map is the same size as the input image.

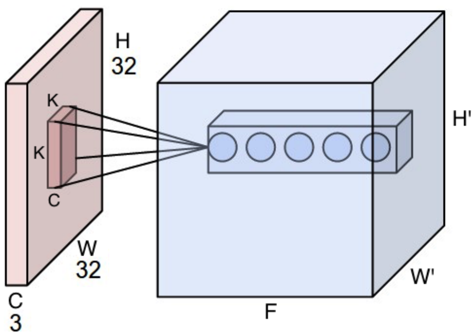


Stride 1 with Padding   Feature Map

- Usually "same" is better since it looks at structures around border.
  - position information is implicitly encoded in the CNN features based on the zero-padding border.

2D Convolutional layer

- **Input:** HxW image with C channels
  - For example, in the first layer, C=3 for RGB channels.
  - defines a 3D volume: C x H x W (or H x W x C)
- **Features:** apply F convolution filters to get F feature maps.
  - Each feature map uses a 3D convolution filter (CxKxK) on the input
  - K is the spatial extent of the filter; total FCKK parameters
- **Activation:**
  - an activation function can be applied before output
- **Output:** a feature map with F channels
  - defines a 3D volume: F x H' x W'
  - H' and W' depend on various factors.

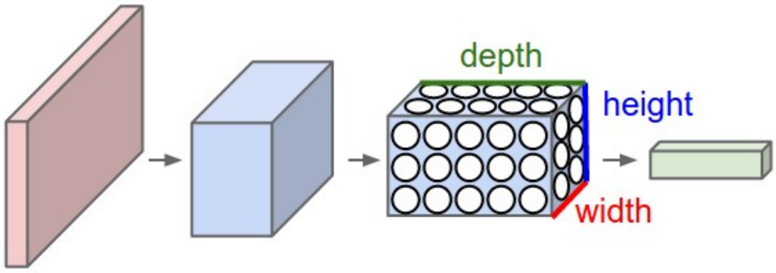


Combining Convolutional Layers

- Concatenate several convolutional layers.

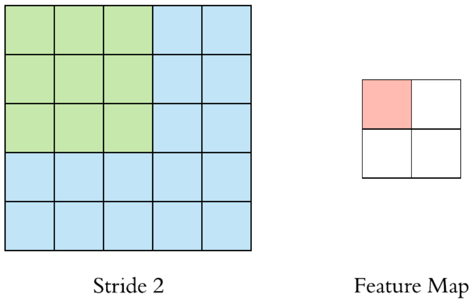
  - From layer to layer
    - spatial resolution decreases
    - number of feature maps increases

- Can extract high-level features in the final layers
- Feature map representation:

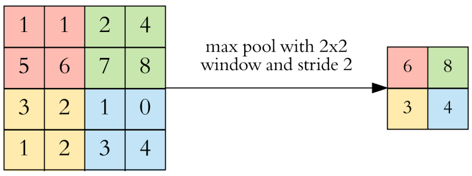


Spatial sub-sampling

- reduce the feature map size by subsampling feature maps between convolutional layers
  - *stride* for convolution filter - step size when moving the windows across the image.



- Spatial sub-sampling
  - *max-pooling layer* - use the maximum over a pooling window
    - gathers features together, summarizes features in local region.

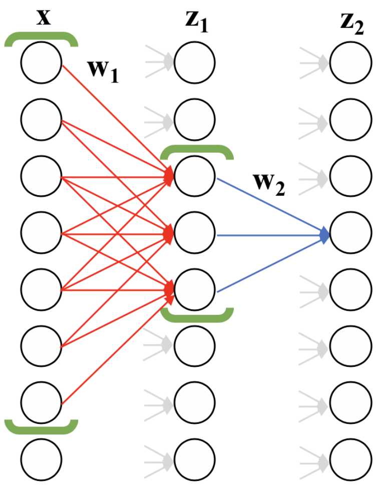


    - introduces **translation invariance**
      - it doesn't matter where the maximal feature is located locally, it is passed to the next layer.
      - increases robustness to small changes in configuration of features

Receptive field size

- Stacking convolutional layers increases the effective size of the pattern filter
  - called **receptive field** - what pixels in the input affect a particular node.
  - larger receptive fields can see larger patterns.
  - Example: 2 convolutional layers

- $|\mathbf{w_1}| = 5, |\mathbf{w_2}| = 3$, receptive field size = $|\mathbf{w_1}| + |\mathbf{w_2}| - 1 = 7$

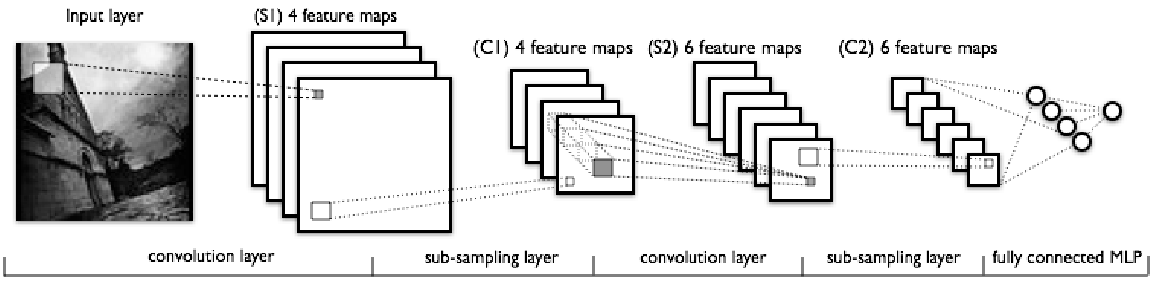


Advantages of Convolution Layers

- The convolutional filters extract the same features throughout the image.
  - Useful because the object can appear in different locations of the image (global translation equivarience).
- Pooling makes it robust to changes in feature configuration (local translation invariance).
- The number of parameters is small compared to Dense (Fully-connected) layer
  - Example: input is C x H x W, and output is F x H x W
    - Number of MLP parameters: (CHW+1) x (FHW)
    - Number of CNN parameters: F x (CKK+1)

Fully-connected layers (MLP)

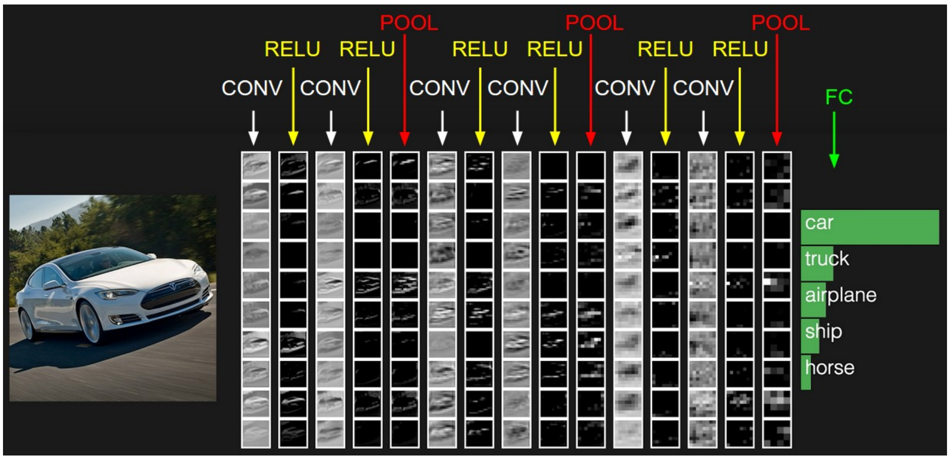
- after several convolutional layers, input the feature map into an MLP to get the final classification.
- also called "fully-connected" (FC) layers.



Example: Object classification CNN

- Each layer shows its feature maps for the example image.
  - early layers extract *low-level* (visual) features

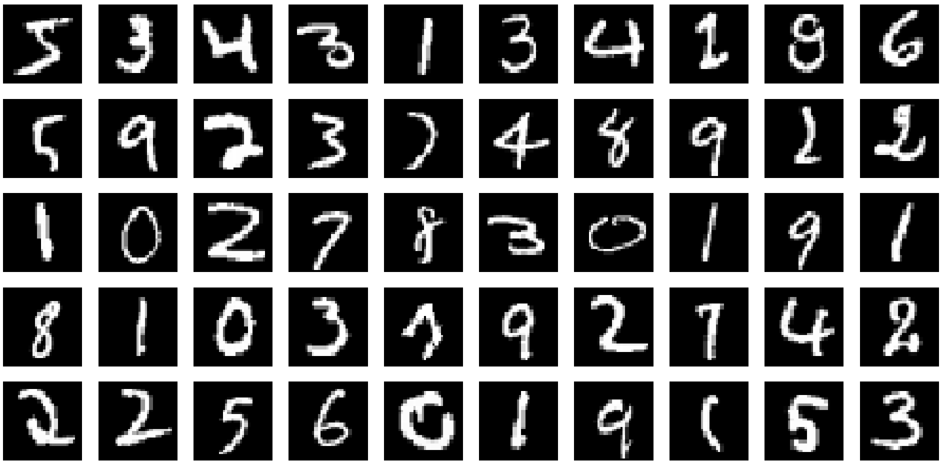
        - e.g., corners, edges
    - middle layers extract *mid-level* (part) features
        - e.g., object parts
    - later layers extract *high-level* (semantic) features.
        - e.g., object



- The number of feature channels increases with each layer
    - combining low-level parts together to get more higher-level parts
        - e.g., {edges, corners} -> {wheel, door, window}
    - trading off spatial resolution for semantic specificity
        - e.g., 512x512x3 RGB image -> 8x8x512 semantic features
- the spatial resolution decreases with each layer
    - increase the window size (receptive field) on the object
    - high-level semantic correspond to large regions.

CNN on MNIST

In [23]:
```
# Example images
plt.figure(figsize=(8,4))
show_imgs(trainimg[0:50])
```



4D tensor format

- There are two common formats for the 4-D tensor:
  - "NCHW" - batch, channel, height, width - called `'channels_first'`
  - "NHWC" - batch, height, width, channel - called `'channels_last'`
- NHWC is required for CPU version of Tensorflow 2.

In [24]:
```
# use keras backend (K) to force channels-last ordering (for CPU compatability)
K.set_image_data_format('channels_last')
```

Example on MNIST

- Pre-processing
  - scale to [0,1]
  - 4-D tensor: (sample, height, width, channel)
    - channel = 1 (grayscale)
  - create training/validation sets

In [25]:
```
# scale to 0-1
trainI = (trainimg.reshape((6000,28,28,1)) / 255.0)
testI  = (testimg.reshape((10000,28,28,1)) / 255.0)
print(trainI.shape)
print(testI.shape)
```

```
(6000, 28, 28, 1)
(10000, 28, 28, 1)
```

- Generate fixed training and validation sets

In [26]:
```
# generate fixed validation set of 10% of the training set
vtrainI, validI, vtrainYb, validYb = \
  model_selection.train_test_split(trainI, trainYb,
  train_size=0.9, test_size=0.1, random_state=4487)

validsetI = (validI, validYb)
```

Shallow CNN Architecture

- 1 Convolution layer
  - 5x5x1 kernel, 10 features, *stride* = 2 (step-size between sliding windows)
  - No pooling here since the image input is small (28x28)
  - Input: 28x28x1 (grayscale image) -> Output: 14x14x10
- 1 fully-connected layer (MLP), 50 nodes
  - Input: 14x14x10=1960 -> Output: 50
- Classification output node

In [28]:
```
# initialize random seed
K.clear_session(); random.seed(4487); tf.random.set_seed(4487)

# build the network
nn = Sequential()
nn.add(Conv2D(10, (5,5), strides=(2,2), # channel, kernel size, stride
              activation='relu', padding='same',  # activation, convolution padding
              input_shape=(28,28,1)))
nn.add(Flatten())   # flatten the feature map into a vector to apply Dense layers
nn.add(Dense(units=50, activation='relu'))
nn.add(Dense(units=10, activation='softmax'))

# compile and fit the network
nn.compile(loss=keras.losses.categorical_crossentropy,
           optimizer=keras.optimizers.SGD(lr=0.02, momentum=0.9, nesterov=True),
           metrics=['accuracy'])
history = nn.fit(vtrainI, vtrainYb, epochs=100, batch_size=50,
                 callbacks=callbacks_list,
                 validation_data=validsetI, verbose=False)
```

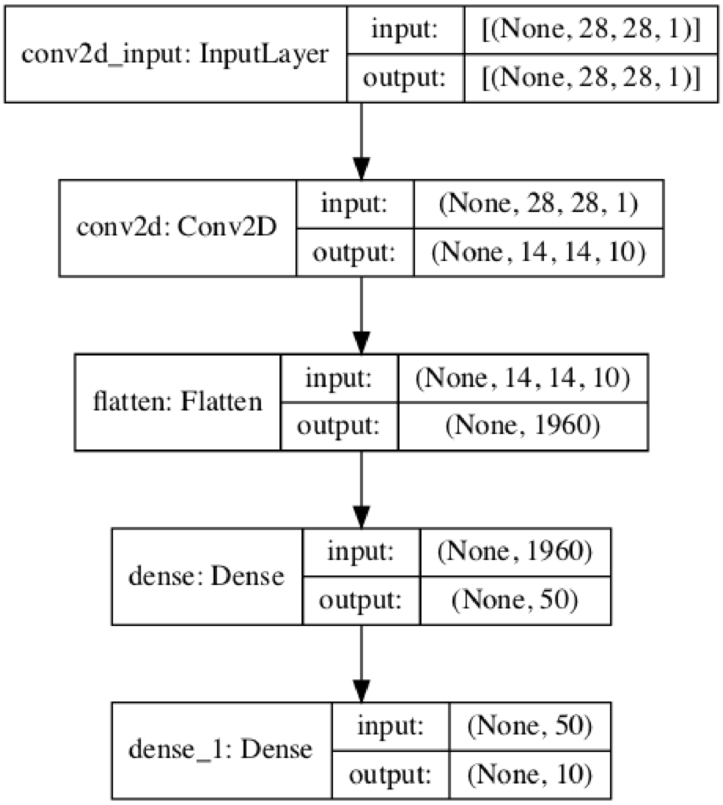
```
Epoch 00012: early stopping
```

In [29]:
```
nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 14, 14, 10)        260
_________________________________________________________________
flatten (Flatten)            (None, 1960)              0
_________________________________________________________________
dense (Dense)                (None, 50)                98050
_________________________________________________________________
dense_1 (Dense)              (None, 10)                510
=================================================================
Total params: 98,820
Trainable params: 98,820
Non-trainable params: 0
_________________________________________________________________
```

In [30]:
```
# visualize the network
tf.keras.utils.plot_model(nn, to_file='tmp_model.png', show_shapes=True)
```

Out[30]:



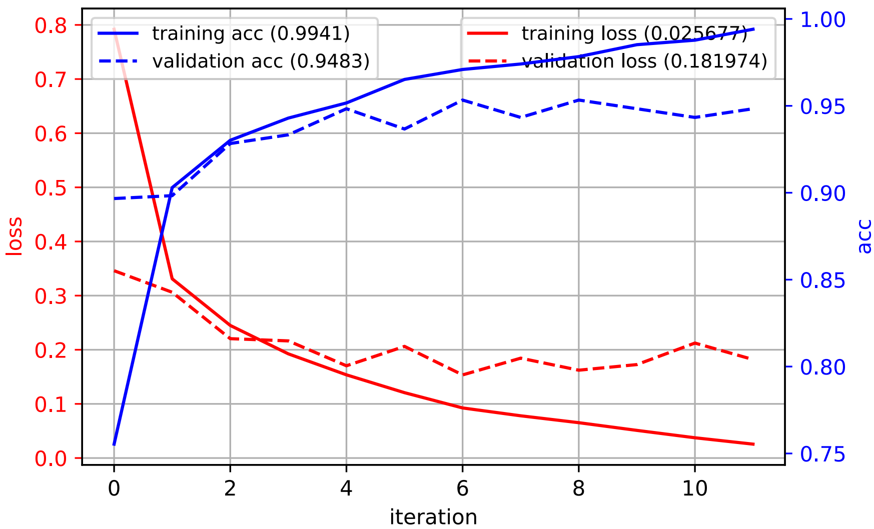
- test results
  - for comparison, the best MLP from Lecture 8 was 0.9499 accuracy
    - 3 layer MLP: (500, 500, 10) with 648,010 parameters

In [31]:

```
plot_history(history)

predY = argmax(nn.predict(testI, verbose=False), axis=-1)
acc = metrics.accuracy_score(testY, predY)
print("test accuracy:", acc)
```

```
test accuracy: 0.9487
```

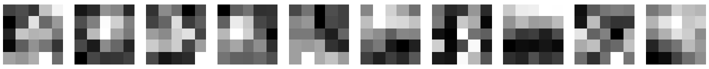


- Visualize the convolutional filters
  - filters are looking for local stroke features
    - corners, edges, lines

In [32]:

```
W = nn.get_layer(index=0).get_weights()[0]
print(W.shape)
flist = [squeeze(W[:,:,:,c]) for c in range(10)]
show_imgs(flist)
```

```
(5, 5, 1, 10)
```



Deep CNN Architecture

- 3 Convolutional layers
  - 5x5x1 kernel, stride 2, 10 features (output 14x14x10)
  - 5x5x10 kernel, stride 2, 40 features (output 7x7x40)
  - 5x5x40 kernel, stride 1, 80 features (output 7x7x80)
    - set stride as 1 to avoid reducing the feature map too much)
- 1 fully-connected layer (7x7x80=3920 -> 50)
- Classification output node

In [33]:

```
# initialize random seed
random.seed(4487); tf.random.set_seed(4487)
# build the network
nn = Sequential()
nn.add(Conv2D(10, (5,5), strides=(2,2), activation='relu',
              padding='same', input_shape=(28,28,1)))
nn.add(Conv2D(40, (5,5), strides=(2,2), activation='relu', padding='same'))
nn.add(Conv2D(80, (5,5), strides=(1,1), activation='relu', padding='same'))
nn.add(Flatten())
nn.add(Dense(units=50, activation='relu'))
nn.add(Dense(units=10, activation='softmax'))

# compile and fit the network
nn.compile(loss=keras.losses.categorical_crossentropy,
           optimizer=keras.optimizers.SGD(lr=0.02, momentum=0.9, nesterov=True),
          metrics=['accuracy'])
history = nn.fit(vtrainI, vtrainYb, epochs=100, batch_size=50,
                 callbacks=callbacks_list,
                 validation_data=validsetI, verbose=False)
```

```
Epoch 00010: early stopping
```

In [34]:

```
nn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 14, 14, 10)        260
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 40)          10040
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 80)          80080
_________________________________________________________________
flatten_1 (Flatten)          (None, 3920)              0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                196050
```

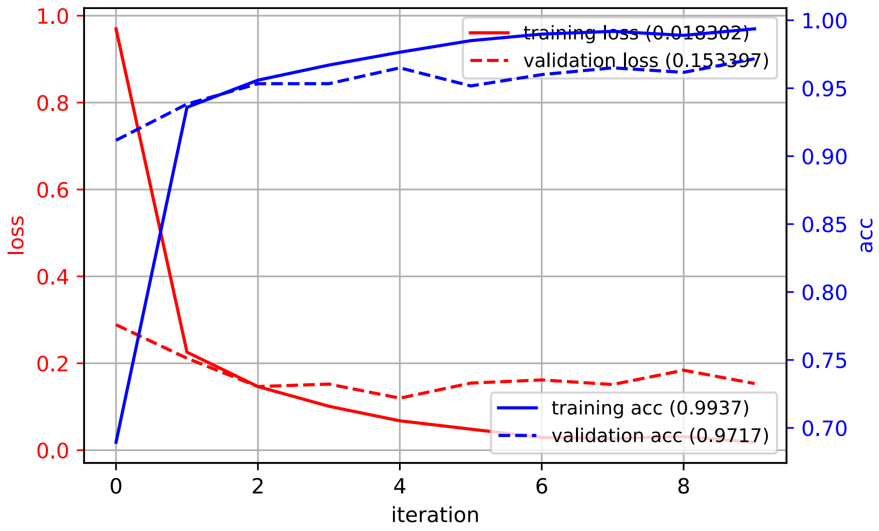
```
dense_3 (Dense)              (None, 10)                510
=================================================================
Total params: 286,940
Trainable params: 286,940
Non-trainable params: 0
_________________________________________________________________
```

In [35]:

```
plot_history(history)

predY = argmax(nn.predict(testI, verbose=False), axis=-1)
acc = metrics.accuracy_score(testY, predY)
print("test accuracy: ", acc)
```

```
test accuracy:  0.9667
```



Summary

- Convolution operation
  - looks at the local structure of the signal (1D, 2D, 3D, etc).
  - two intepretations: filtering, pattern matching
- Convolutional neural network (CNN)
  - convolutional layer - use convolution instead of dense connections
  - learns to extract image features, and learns classifier.