Traditional pattern recognition models use hand-crafted features and relatively simple trainable classifiers. This approach has the following limitations:

- It is very tedious and costly to develop hand-crafted features.
- The hand-crafted features are usually highly dependent on one application, and cannot be transferred easily to other applications.
Smaller Network: CNN

- We know it is good to learn a small model.
- From this fully connected model, do we really need all the edges?
- Can some of these be shared?
Consider learning an image:

- Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters

“beak” detector
Same pattern appears in different places: They can be compressed!

What about training a lot of such “small” detectors and each detector must “move around”.

They can be compressed to the same parameters.
A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.
Convolution

These are the network parameters to be learned.

Each filter detects a small pattern (3 x 3).

6 x 6 image
### Convolution

**6 x 6 image**

**Filter 1**

```
1 -1 -1
-1 1 -1
-1 -1 1
```

**stride=1**

- **Dot product**: 3, -1
Convolution

If stride=2

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 \\
\end{bmatrix}$$

6 x 6 image

Filter 1

$$\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1 \\
\end{bmatrix}$$

3, -3
Convolution

stride=1

6 x 6 image

Filter 1
Convolution

stride=1

6 x 6 image

Filter 2

Repeat this for each filter

Two 4 x 4 images
Forming 2 x 4 x 4 matrix
Color image: RGB 3 channels

Color image

Filter 1

Filter 2

1 0 0 0 0 0 1
0 1 0 0 1 0 0
0 0 1 1 0 0 0
1 0 0 0 1 0 0
0 1 0 0 1 0 0
0 0 1 0 1 0 0
0 0 1 0 1 0 0
Convolution v.s. Fully Connected

image

convolution

Fully-connected
6 x 6 image

fewer parameters!

Only connect to 9 inputs, not fully connected
Filter 1

6 x 6 image

Fewer parameters

Even fewer parameters

Shared weights
The whole CNN

cat dog ……

Fully Connected Feedforward network

Convolution
Max Pooling
Convolution
Max Pooling

Can repeat many times

Flattened
Max Pooling

Filter 1

Filter 2

-1 -1 -1
-1 1 -1
-1 1 1

-3 -1
-3 1
-3 -3
3 -2

3 -1
0 1
-2 -1

-1 -1
-1 -1
-2 1
-4 3
Why Pooling

- Subsampling pixels will not change the object

We can subsample the pixels to make image smaller, fewer parameters to characterize the image.
A CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity
Max Pooling

6 x 6 image

New image but smaller

Each filter is a channel
The whole CNN

A new image

Smaller than the original image

The number of channels is the number of filters

Can repeat many times

Convolution

Max Pooling

Convolution

Max Pooling
The whole CNN

cat dog ……

Fully Connected Feedforward network

Convolution
Max Pooling
Convolution
Max Pooling

A new image
A new image

Flattened
Flattening

-1 1
0 3

Flattened

3 0 1
1 3 -1
0 -1 1
0 3

Fully Connected Feedforward network
Only modified the network structure and input format (vector -> 3-D tensor)

```
model2.add(Convolution2D( 25, 3, 3, input_shape=(28, 28, 1) ))
```

There are 25 3x3 filters.

```
model2.add(MaxPooling2D((2,2)))
```

Input_shape = ( 28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB
CNN in Keras

Only modified the network structure and input format (vector -> 3-D array)

```
model2.add(Convolution2D( 25, 3, 3, input_shape=(28,28,1) )
```

How many parameters for each filter?

9

```
model2.add(MaxPooling2D((2,2))
```

```
model2.add(Convolution2D(50, 3, 3))
```

How many parameters for each filter?

225 = 25x9

```
model2.add(MaxPooling2D((2,2))
```

```
model2.add(Convolution2D(50, 3, 3))
```

```
model2.add(MaxPooling2D((2,2))
```

```
model2.add(Convolution2D(50, 3, 3))
```

Input

Convolution

Max Pooling

Convolution

Max Pooling

How many parameters for each filter?

9

225 = 25x9

50 x 11 x 11

50 x 5 x 5
Only modified the **network structure** and input format (vector -> 3-D array)

**CNN in Keras**

Input

- 1 x 28 x 28
- Convolution
- 25 x 26 x 26
- Max Pooling
- 25 x 13 x 13
- Convolution
- 50 x 11 x 11
- Max Pooling
- 50 x 5 x 5

Output

- Flattened
- 1250
- Fully connected feedforward network

```python
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))
model2.add(Flatten())
```
AlphaGo

19 x 19 matrix
Black: 1
white: -1
none: 0

Fully-connected feedforward network can be used
But CNN performs much much better
The following is quotation from their Nature article:

**Neural network architecture.** The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a $23 \times 23$ image, then convolves $k$ filters of kernel size $5 \times 5$ with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a $21 \times 21$ image, then convolves $k$ filters of kernel size $3 \times 3$ with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size $1 \times 1$ with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128, 256$ and 384 filters.
CNN in speech recognition

The filters move in the frequency direction.
CNN in text classification

Source of image:
A CNN for MNIST in Keras: accuracy > 0.99!

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))