

# Convolutional Neural Networks

Liangliang Cao

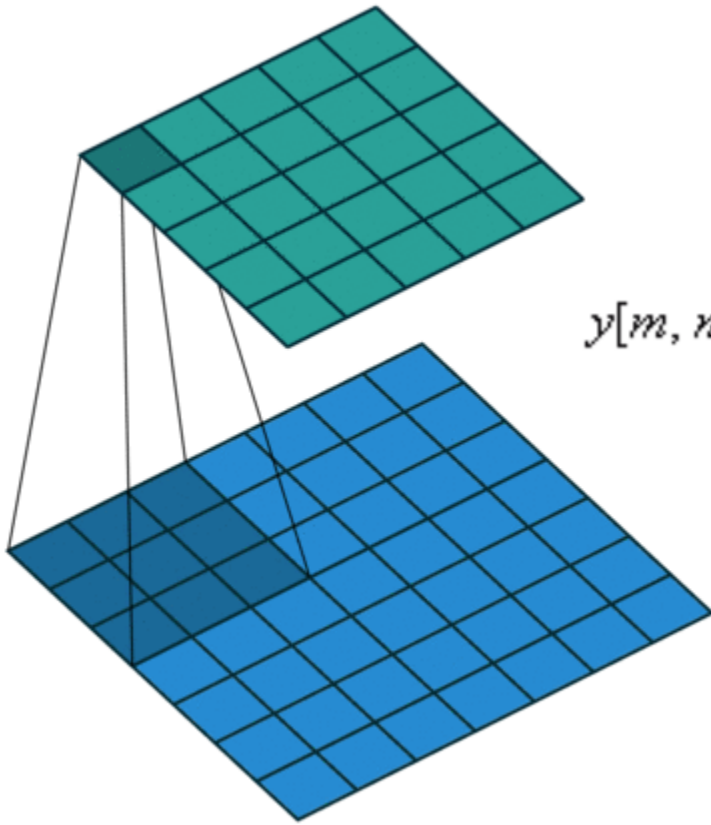
<https://columbia6894.github.io/>

# Outline

- Discussing conv. filters from traditional viewpoints
- The first popular deep CNN: LeNet in 1998
- The second popular deep CNN: AlexNet in 2012
- Why 14 years? Challenges of implementing AlexNet?
- Improving CNNs
  - 1x1 convolution
  - Residual network

# Convolutional Filters

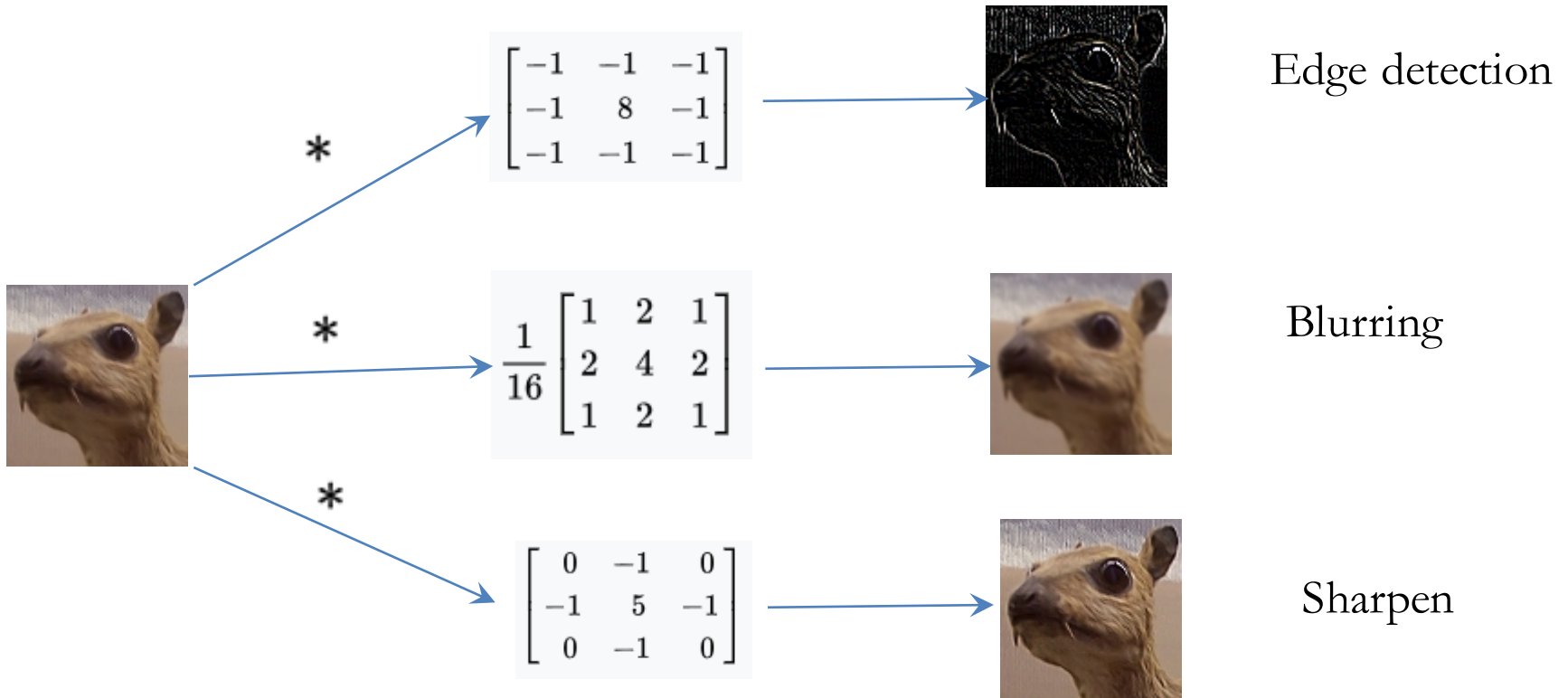
- Image filtering are usually represented by the convolution between an image and a mask.



$$y[m, n] = x[m, n] * h[m, n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i, j] \cdot h[m-i, n-j]$$

# Image Filters

- Image filtering are usually represented by the convolution between an image and a mask.



# Discussions

- Filters are powerful for many vision applications

We can use filters for recognition, enhancement...

That is why nowadays CNNs almost dominate all vision applications

# Discussions

- Filters are powerful for many vision applications
- Convolutions are expensive
  - At every pixel we need do multi-multiplication with its neighborhood values
  - Algorithms of speedup\*: integral image, separable filters, time domain convolution -> frequency multiplication, etc
  - Hardware of speedup: GPU, TPU

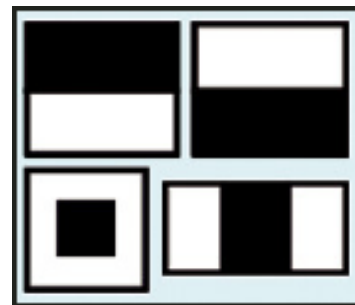
\*This suggests a number of research ideas of improving deep cnn

# Discussions

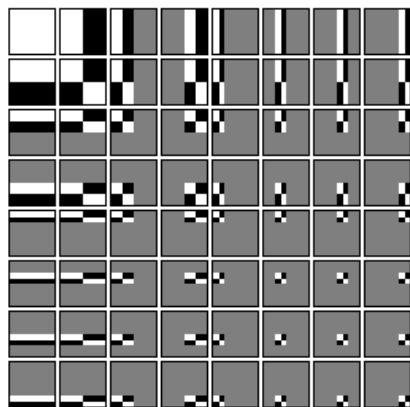
- Filters are powerful for many vision applications
- Convolutions are expensive
- How many filters can we learn?
  - Dozens? Hundreds? Millions? More?

# Huge Amount of Filters: An Example

[Viola and Jones]: face detection via millions\* of simple filters



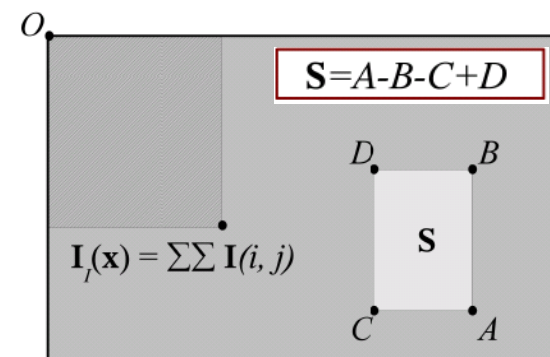
*Haar Wavelet*



*Haar like features*

*Given two adjacent rectangular regions, sums up the pixel intensities in each region and calculates the difference between the two sums*

*Efficient computation*



\*This suggests to find ways to train numerous filters...



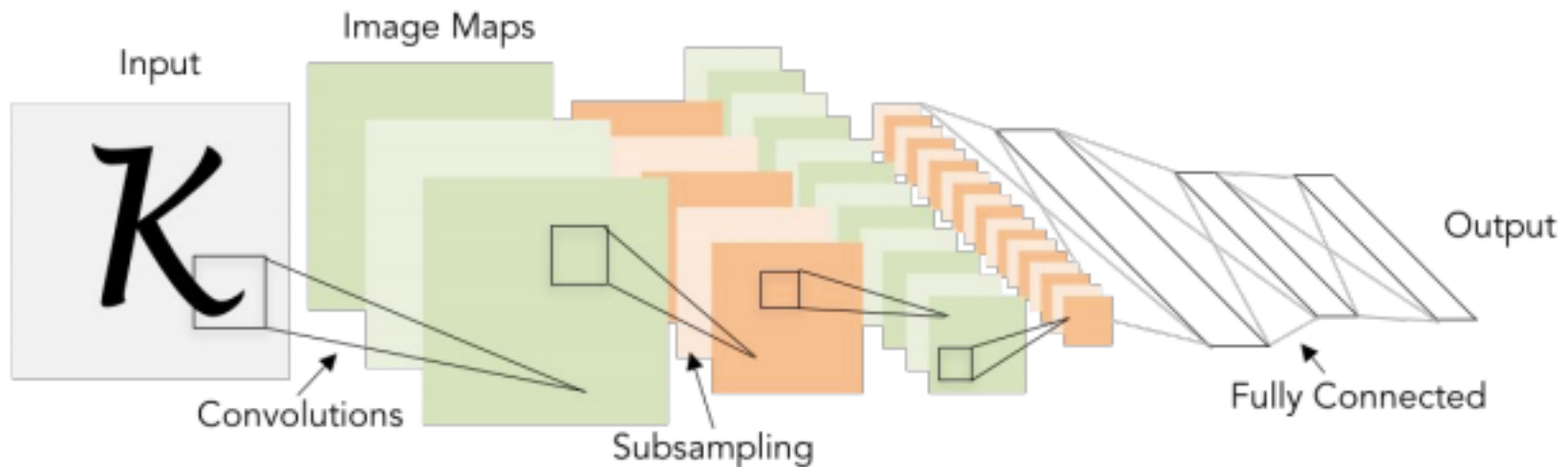
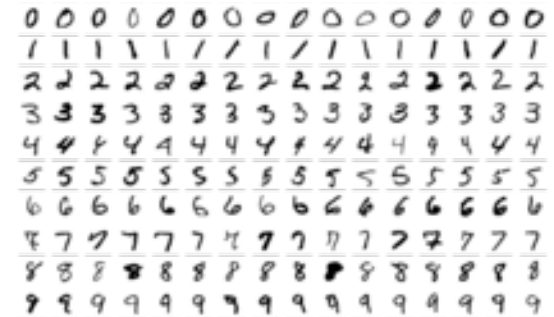
# Discussions

- Filters are powerful for many vision applications
- Convolutions are expensive
- How many filters can we learn?
- How to manage larger neighborhood?
  - Sub-sample the image
  - Larger receptive fields (i.e., filter size)
  - Stack multi convolutional layers together -> deep CNNs

**Let's Go to Multi-Layer CNNs  
(deep CNNs)!**

# The First Popular Deep CNN

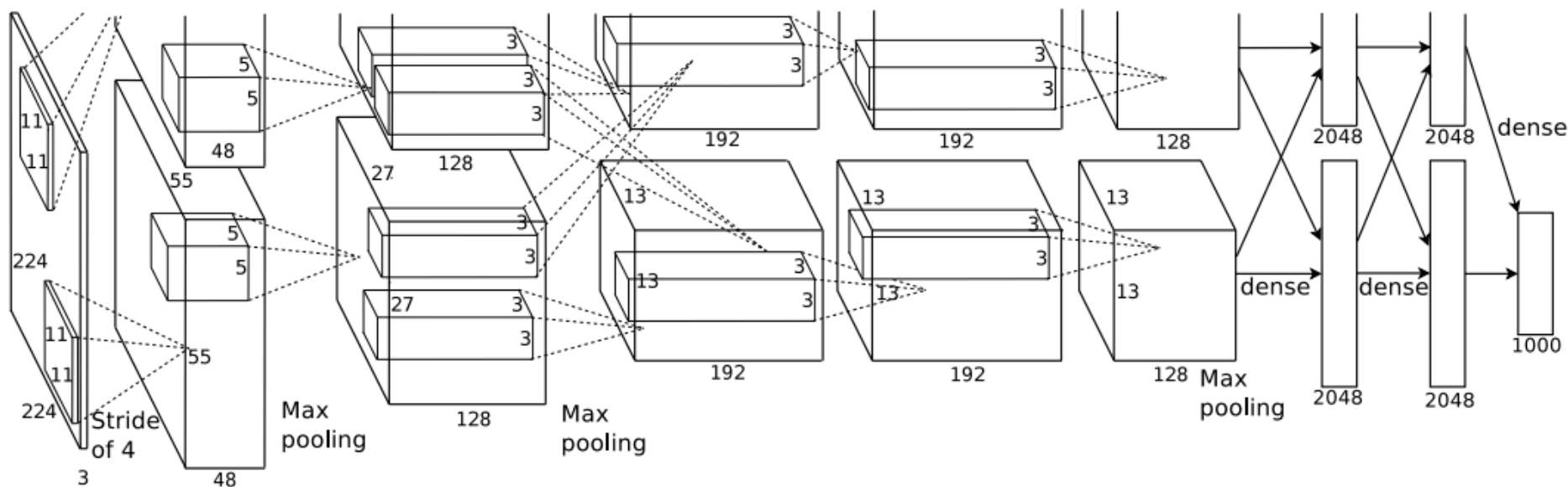
- LeCun, Bottou, Bengio, Haffner, Gradient-based learning applied to document recognition, Proc. IEEE, 1998



# The Second Popular Deep CNN

- Krizhevsky, Sutskever, Hinton,  
ImageNet Classification with Deep  
Convolutional Neural Networks,  
NIPS 2012

IMAGENET



# Why Took 14 years? (1998-2012)

- People do not trust local minimum and may be annoyed by SGD failures.

Which of the following will fail CNNs on MNIST?

- Use the raw pixel values between  $[0, 255]$
- Initialize all the CNN weights as 0
- Use no intercept (i.e.,  $Wx$  instead of  $Wx+b$ ) in the fully connect layer
- The batch size is too small (i.e., one sample per batch)
- Use the whole dataset as one batch
- Do not shuffle the data before training

# Why Took 14 years? (1998-2012)

- People do not trust local minimum and may be annoyed by SGD failures.

Which of the following will fail CNNs on MNIST?

- Use the raw pixel values between  $[0, 255]$

Yes. Almost all CNNs prefer to normalize pixel value normalized between  $[0,1]$

# Why Took 14 years? (1998-2012)

- People do not trust local minimum and may be annoyed by SGD failures.

Which of the following will fail CNNs on MNIST?

- Initialize all the CNN weights as 0

Yes. network weights should be initialized randomly

# Why Took 14 years? (1998-2012)

- People do not trust local minimum and may be annoyed by SGD failures.

Which of the following will fail CNNs on MNIST?

- Use no intercept (i.e.,  $Wx$  instead of  $Wx+b$ ) in the fully connect layer

No. Network with zero intercepts will still work.



# Why Took 14 years? (1998-2012)

- People do not trust local minimum and may be annoyed by SGD failures.

Which of the following will fail CNNs on MNIST?

- The batch size is too small (i.e., one sample per batch)

No. Small batch size will still work, but make the optimization smaller

# Why Took 14 years? (1998-2012)

- People do not trust local minimum and may be annoyed by SGD failures.

Which of the following will fail CNNs on MNIST?

- Use the whole dataset as one batch

Yes. We will lose the “stochastic” factor by taking whole dataset as one batch, and the optimization will fall into bad local minimum.

# Why Took 14 years? (1998-2012)

- People do not trust local minimum and may be annoyed by SGD failures.

Which of the following will fail CNNs on MNIST?

- Do not shuffle the data before training

Yes. Random shuffling is important.

# Why Took 14 years? (1998-2012)

- People do not trust local minimum and may be annoyed by SGD failures.

Which of the following will fail CNNs on MNIST?

- **Use the raw pixel values between [0, 255]**
- **Initialize all the CNN weights as 0**
- Use no intercept (i.e.,  $Wx$  instead of  $Wx+b$ ) in the fully connect layer
- The batch size is too small (i.e., one sample per batch)
- **Use the whole dataset as one batch**
- **Do not shuffle the data before training**

# Why Took 14 years? (1998-2012)

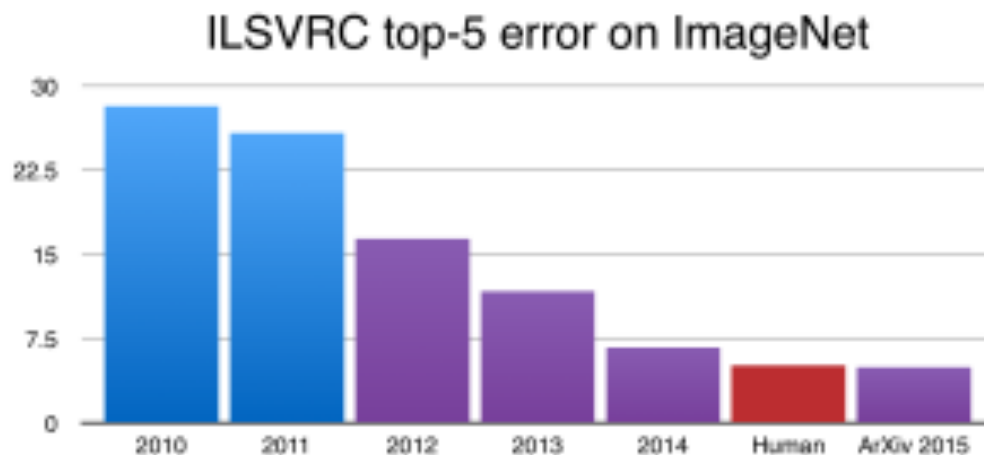
- People do not trust local minimum and may be annoyed by SGD failures.
- On MNIST CNN is not significant better than others

Model	Testing Error
KNN, subsample 16 x 16	1.1%
Boosted tree	1.53%
Non-linear SVM by LeCun'98	1.0%
Non-linear SVM by DeCoste'02	0.56%
2-layer MLP	2.45%
CNN LeNet-5	0.95%

Results from <http://yann.lecun.com/exdb/mnist/>

# Why Took 14 years? (1998-2012)

- People do not trust local minimum and may be annoyed by SGD failures.
- On MNIST CNN is not significant better than others
- But on ImageNet things changed!



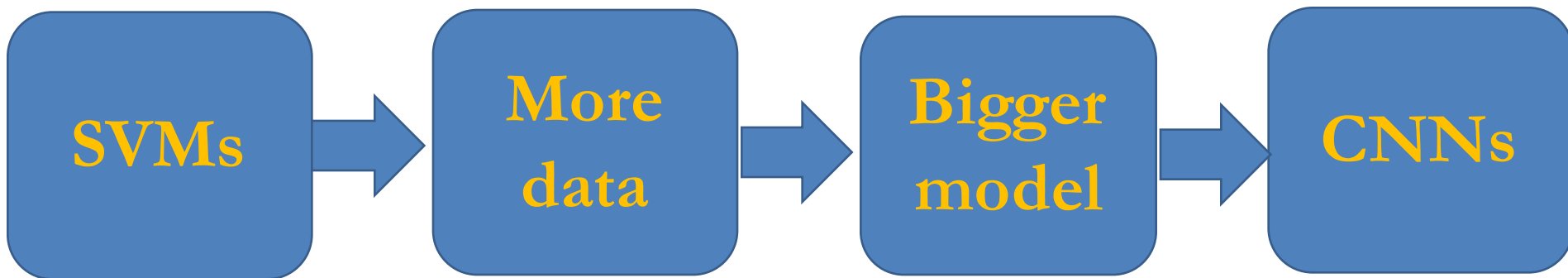
# Differences between MNIST and ImageNet

	MNIST	ImageNet LSVRC
Image size	28 x 28 x 1	224 x 224 x 3*
Num of images	60K	1,200K
Num of category	10	1000
In-class variation	small	large

\*Resized size. Can be as large as 512 x 512

# Differences between MNIST and ImageNet

	MNIST	ImageNet LSVRC
Image size	28 x 28 x 1	224 x 224 x 3
Num of images	60K	1,200K
Num of category	10	1000
In-class variation	small	large





**Let's implement these two popular models.**

## **To implement LeNet is easy ...**

1. Download MNIST data and load them into memory
2. Build a 5 layer CNN model
3. Train model and evaluate

You can even run on your laptop without GPU

# Implement LeNet-5 using Keras

```
model = Sequential()
model.add(Conv2D(filters = 6, kernel_size = 5, strides = 1, activation = 'relu',
                 input_shape = (32,32,1)))
model.add(MaxPooling2D(pool_size = 2, strides = 2))
model.add(Conv2D(filters = 16, kernel_size = 5, strides = 1, activation = 'relu',
                 input_shape = (14,14,6)))
model.add(MaxPooling2D(pool_size = 2, strides = 2))
model.add(Flatten())
model.add(Dense(units = 120, activation = 'relu'))
model.add(Dense(units = 84, activation = 'relu'))
model.add(Dense(units = 10, activation = 'softmax'))
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics
              = ['accuracy'])
model.fit(X_train ,Y_train, steps_per_epoch = 10, epochs = 40)
```

# Explain LeNet-5

- filters
- kernel\_size
- Strides
- pool\_size
- model.add(Flatten())
- activation=relu/sigmoid/softmax

## But to implement AlexNet is hard...



Alex Krizhevsky was working on CNNs in 2011. He recalled:

“Ilya convinced me that with **an additional week** of effort, we could get equally good results on ImageNet. It actually took **five months** to match the 2010 state-of-the-art, and **several more months** to improve on it convincingly.”

“Time scales aside, his intuition was correct.”

## **But to implement AlexNet is hard...**

Suppose you are the chief architect, what is the solution for

- load 1.2M images into memory
- do convolution via GPUs
- AlexNet model: two stream using 2 GPUs (not necessary though)

# Challenge 1: Cannot Load All ImageNet Data into Memory

- Can not load into memory:  $1.2\text{M} \times 224 \times 224 \times 3 = 180\text{G}$
- Keras' solution: use data iterator

```
class NaiveImageNetIterator:
```

```
    def __init__(self, total_batches):
```

```
        self.ib, self.nb = 0, total_batches
```

```
    def __iter__(self):
```

```
        return self
```

```
    def next(self): # Python 3: def __next__(self)
```

```
        if self.ib >= self.nb: raise StopIteration
```

```
        else:
```

```
            self.ib += 1
```

```
            return Load_Batch_from_Disk(self.ib)
```

# Challenge 1: Cannot Load All ImageNet Data into Memory

Can not directly load into mem:  $1.2\text{M} \times 224 \times 224 \times 3 = 180\text{G}$

- Keras' solution: use data iterator

```
class NaiveImageNetIterator: ....
```

```
data_iterator = NaiveImageNetIterator(120)
```

```
model.fit_generator(data_iterator, sample_per_epoch=1000)
```



# Challenge 1: Cannot Load All ImageNet Data into Memory

Can not directly load into mem:  $1.2\text{M} \times 224 \times 224 \times 3 = 180\text{G}$

- Keras' solution: use data iterator
- Tensorflow's low level API: use `tf.data.Dataset`
  - `tf.data.Dataset` can generate an iterator of Tensor objects

[https://www.tensorflow.org/api\\_docs/python/tf/data](https://www.tensorflow.org/api_docs/python/tf/data)

- Many detection toolkits use TFRecord to organize many images

- `tensorpack` provides an efficient & easy to use 3<sup>rd</sup> party implementation

<https://github.com/tensorpack/tensorpack>

# Challenge 1: Cannot Load All ImageNet Data into Memory

Can not directly load into mem:  $1.2\text{M} \times 224 \times 224 \times 3 = 180\text{G}$

- Keras' solution: use data iterator
- Tensorflow's native solution: use `tf.data.Dataset`
- 3<sup>rd</sup> Party implementation: Tensorpack
  - (<https://github.com/tensorpack/tensorpack>)
    - Use `Tensorpack.dataflow`
    - See example: `ImageNetModels/imagenet_utils.py`
    - The most efficient solution so far

I may provide a note with more details after the class.

But you may have to dig into these examples to play with these solutions

## Challenge 2: Convolution via GPUs

Convolution in GPU is not trivial

- Multi-channel (traditional CV do single channel)
- Multi kernel size (optimization of 5x5 filter differs from 7x7)

See Alex's dizzying code

<https://code.google.com/archive/p/cuda-convnet/>

## Challenge 2: Convolution via GPUs

Convolution in GPU is not trivial

- Multi-channel (traditional CV do single channel)
- Multi kernel size (optimization of 5x5 filter differs from 7x7)

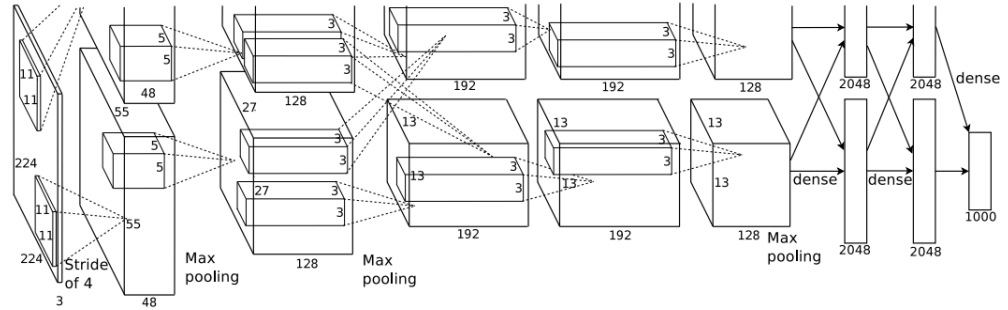
Use NVida's library:

- cuBLAS in early days (converting conv to matrix multiply)
- cuDNN: Nvidia's dominant weapon in GPU market

# Challenge 3: Two Stream CNN

Amazing hacks in 2012

No longer necessary with the  
new GPU cards



# Implement AlexNet with Keras

# layer 1

```
alexnet.add(Conv2D(96, (11, 11),  
    input_shape=img_shape,  
    padding='valid',  
    kernel_regularizer=l2(l2_reg)))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('relu'))  
alexnet.add(MaxPooling2D(strides=(  
    4, 4)))
```

# layer 2

```
alexnet.add(Conv2D(256, (5, 5),  
    padding='same'))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('relu'))  
alexnet.add(MaxPooling2D(pool_size=(2, 2)))
```

What is the number of para. in Layer 1

- $(11 \times 11 \times 3) * 96 = 35K$

What is the output size of layer 1?

- $(224-11) / 4 + 1 = 55$
- Output size  $(56 \times 56 \times 96)$

What is the number of para in layer 2?

- $(5 \times 5 \times 96) * 256 = 710K$

What is the output size of layer 2?

- $55/2 = 27$
- Output size  $(27 \times 27 \times 256)$

# Implement AlexNet with Keras

# layer 1

```
alexnet.add(Conv2D(96, (11, 11),  
    input_shape=img_shape,  
    padding='same',  
    kernel_regularizer=l2(l2_reg)))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('relu'))  
alexnet.add(MaxPooling2D(pool_  
    size=(2, 2)))
```

# layer 2

```
alexnet.add(Conv2D(256, (5, 5),  
    padding='same'))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('relu'))  
alexnet.add(MaxPooling2D(pool_siz  
    e=(2, 2)))
```

# layer 3

```
alexnet.add(ZeroPadding2D((1, 1)))  
alexnet.add(Conv2D(512, (3, 3),  
    padding='same'))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('relu'))  
alexnet.add(MaxPooling2D(pool_siz  
    e=(2, 2)))
```

# layer 4

```
alexnet.add(ZeroPadding2D((1, 1)))  
alexnet.add(Conv2D(1024, (3, 3),  
    padding='same'))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('relu'))
```

# Implement AlexNet in Keras (con't)

# layer 5

```
alexnet.add(ZeroPadding2D((1, 1)))  
alexnet.add(Conv2D(1024, (3, 3),  
    padding='same'))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('relu'))  
alexnet.add(MaxPooling2D(pool_size=(2,  
    2)))
```

# layer 6

```
alexnet.add(Flatten())  
alexnet.add(Dense(3072))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('relu'))  
alexnet.add(Dropout(0.5))
```

# layer 7

```
alexnet.add(Dense(4096))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('relu'))  
alexnet.add(Dropout(0.5)) #
```

# layer 8

```
alexnet.add(Dense(n_classes))  
alexnet.add(BatchNormalization())  
alexnet.add(Activation('softmax'))
```



# From Keras to TF Estimator

Keras is easy to use but not efficient

- Large memory consumption
- Difficulty to scale to multiple GPUs

Use tensorflow's estimator for large datasets:

- TF Estimator can use Keras' layers
- TF Estimator can replace Keras Sequential() model in large scale

# Improving AlexNet

Try smaller receptive fields, more filters, with more layers

- Matt Zeiler Network
- VggNet

Concatenate multiple size of filters

- GoogLeNet

Two techniques are important:

- 1x1 conv (aka “network in network”)
- Residual Network

# 1x1 convolution

Consider two layers of CNN

- Input:  $56 \times 56 \times 3$
- Layer A:  $(11 \times 11) * 96$  filters, output  $(56 \times 56 \times 96)$ ,
- Layer B:  $(5 \times 5) * 256$  filters output  $(56 \times 56 \times 256)$

Layer B has  $(5 \times 5 \times 96) * 256$  parameters, also consumes a lot of GPU memory. How to reduce the parameter?

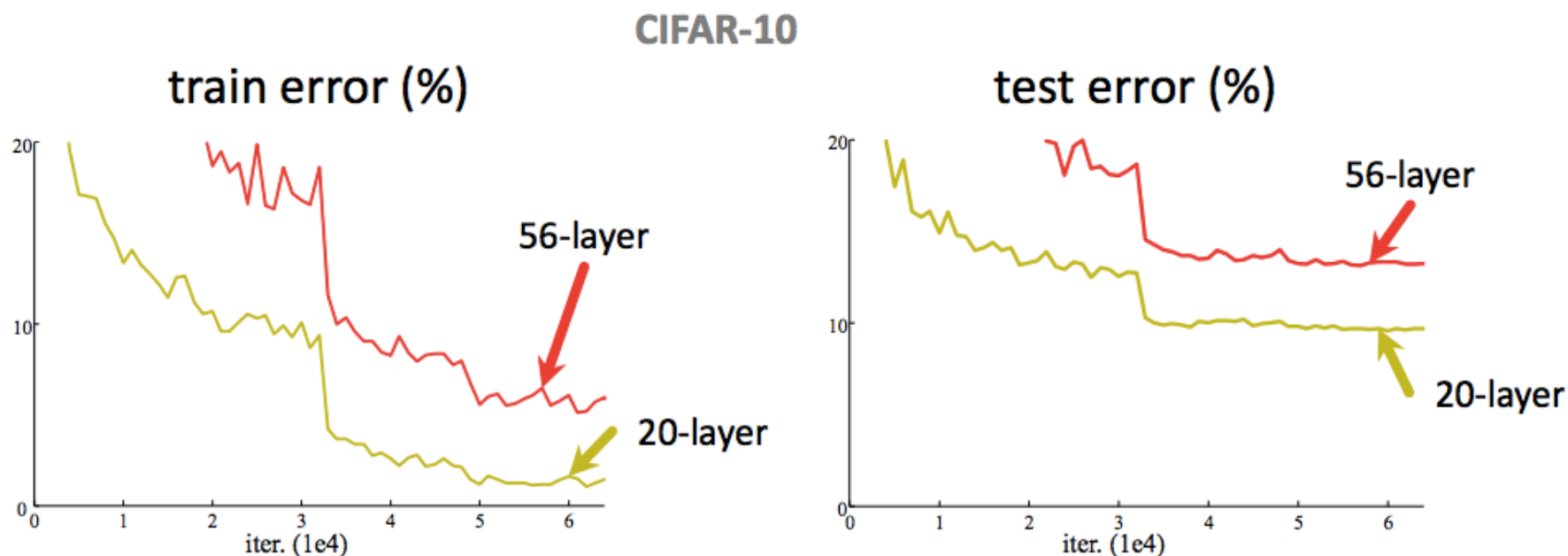
Add a new layer between A and B

- Layer A':  $(1 \times 1) * 32$  filters, output  $(56 \times 56 \times 32)$

Now layer B has  $(5 \times 5 \times 32) * 256$  filters. 3x less parameters!

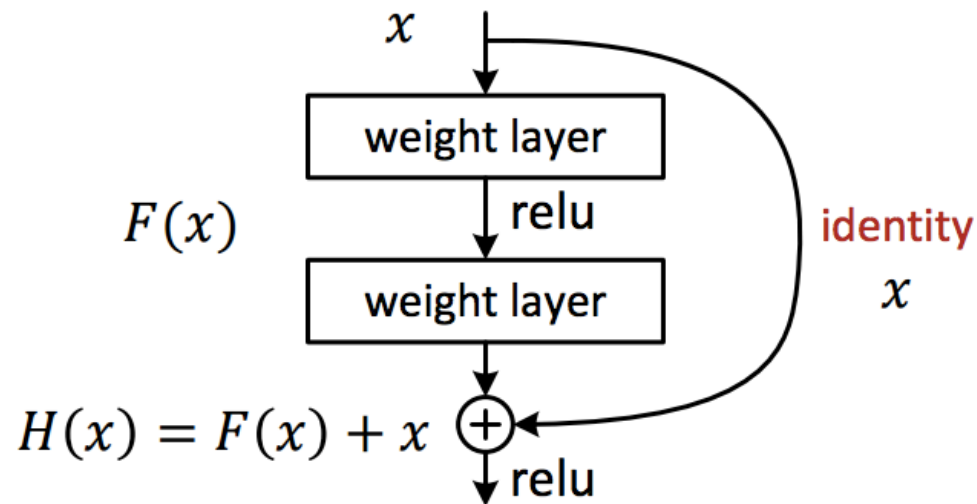
# Why Residual Network?

Problem: Is learning better networks as simple as stacking more layers?



Deep network + residual learning can solve this problem.

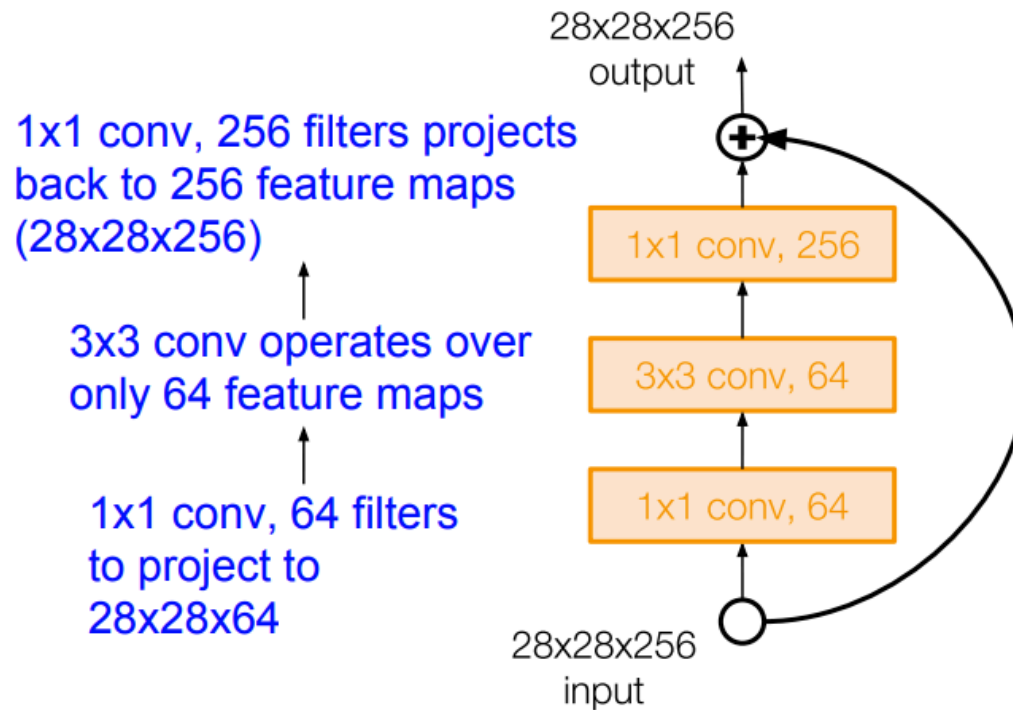
# Residual Net



```
from keras.layers import Conv2D, Input

# input tensor for a 3-channel 256x256 image
x = Input(shape=(3, 256, 256))
# 3x3 conv with 3 output channels (same as input channels)
y = Conv2D(3, (3, 3), padding='same')(x)
# this returns x + y.
z = keras.layers.add([x, y])
```

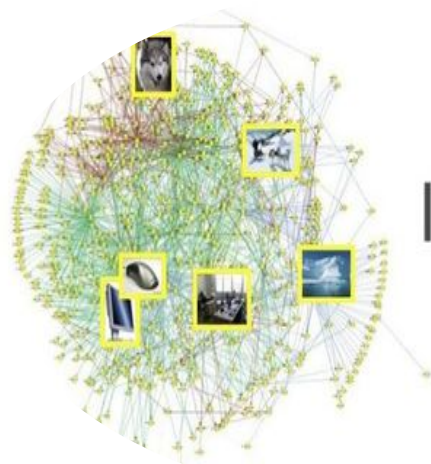
# Residual Net With Bottleneck Structure



A number of future improvement

# Treasure from ImageNet Dataset

By adapting models trained from ImageNet, we can build a decent classifier with limited data.



IMAGENET

Example code :

[http://caffe.berkeleyvision.org/gathered/examples/finetune\\_flickr\\_style.html](http://caffe.berkeleyvision.org/gathered/examples/finetune_flickr_style.html)

Very few new labels

- Tune the last layer
- Or last layer as feature for SVM

Enough new labels

- Tune the whole network

**New tasks**

# But ImageNet May NOT Be Ideal For Course Projects

- Too crowded in the competition
- Relatively difficulty to find novel ideas

If you want to try a final project on large scale recognition, we recommend Celebrity1M faces

After break (8:30pm), will join us our guest lecture Dr. Lei Zhang, who is the creator of Microsoft Celebrity1M.