experience on mp1&mp2

Yihui He

I'm international exchange student
CS 2nd year undergrad
Xi'an Jiaotong University, China

yihuihe@foxmail.com

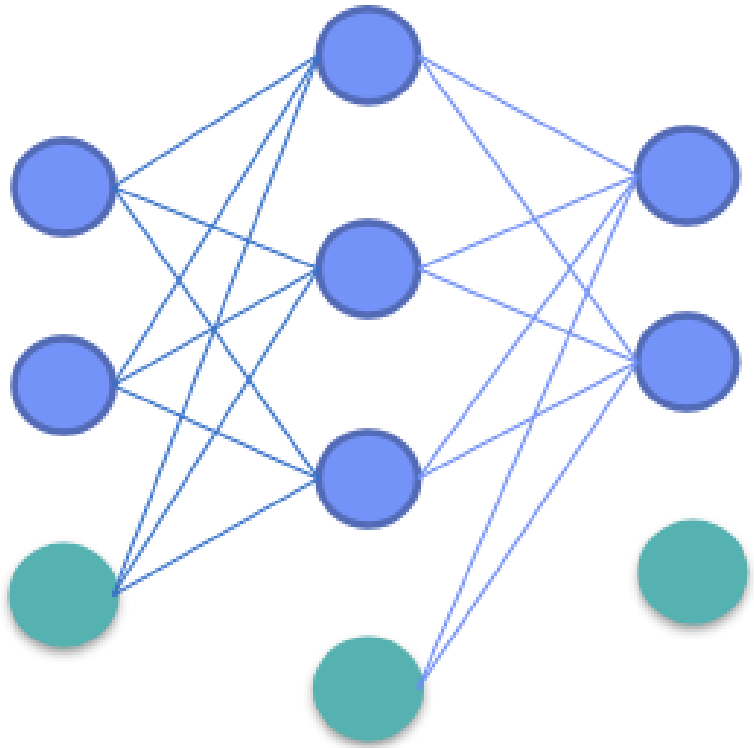
May 18, 2016

Overview

1. mp1
   - tricks
   - new model
2. mp2
   - tricks
   - choosing from different models
   - delving into one model

Goal

- Input: CIFAR 10 image
- Architecture: two-layer neural network
- Output:prediction among 10 classes



tuning hyperparameters

- determine ralation[1] between parameter and backpropagation error: linear, $\theta \propto \delta$ or exponential, $log(\theta) \propto \delta$
- run a grid search(or random search) on a small part of our big dataset

```
for hidden_neurons in range(150,600,50):
  for learning_rate in [1e-3*10**i for i in range(-2,3)
    ]:
    for norm in [0.5*10**i for i in range(-3,3)]:
      [loss_history,accuracy]=\
      train(small_dataset,
            hidden_neurons,learning_rate,norm)
      # dump loss, accuracy history for each setting
      # append highest accuracy of each setting to a .
    csv
```

[1] stanford cs231n

Choosing number of hidden neurons

Table: top accuracy

hidden neurons	learning rate	regularization strength	validation accuracy
350	0.001	0.05	0.516
400	0.001	0.005	0.509
250	0.001	0.0005	0.505
250	0.001	0.05	0.501
150	0.001	0.005	0.5
500	0.001	0.05	0.5

Update methods affect converge rate

1000 iterations, batch size 100

Table: Differences between update methods

accuracy	Train	Validation	Test
SGD	.27	.28	.28
Momentum	.49	.472	.458
Nesterov	.471	.452	.461
RMSprop	.477	.458	.475

These update methods can't make final accuracy higher(sometimes even lower than fine-tuned SGD), but make training much faster.

dropout

Accuracy improves about 3%.
Only need to change one line in code:

```
a2=np.maximum(X.dot(W1)+b1,0)
a2*=(np.random.randn(*a2.shape)<p)/p #add this line
scores=a2.dot(W2)+b2
```

p : dropout rate (usually choosen from .3 .5 .7)
a2 : activation in the second layer.

initialization methods

Three comment initialization for fully connected layer:

- $\mathrm{N}(0,1)\sqrt{1/n}$
- $\mathrm{N}(0,1)\sqrt{2/(n_{in}+n_{out})}$
- $\mathrm{N}(0,1)\sqrt{2/n}$

Significance can't be seen from our two layers shallow neural net. However, initialization is super important in mp2(deep neural net).

questions about these tricks?

new model

After using tricks we mentioned, accuracy is around 55%, neural network architecture is already fixed.

how do we improve accuracy?

algoritms leaderboard[2]

At the very bottom of leaderboard(State-of-the-art is 96%):

83.96%	Discriminative Learning of Sum-Product Networks	NIPS 2012
82.9%	Stable and Efficient Representation Learning with Nonnegativity Constraints	ICML 2014
82.2%	Learning Invariant Representations with Local Transformations	ICML 2012
82.18%	Convolutional Kernel Networks	arXiv 2014
82%	Discriminative Unsupervised Feature Learning with Convolutional Neural Networks	NIPS 2014
80.02%	Learning Smooth Pooling Regions for Visual Recognition	BMVC 2013
80%	Object Recognition with Hierarchical Kernel Descriptors	CVPR 2011
79.7%	Learning with Recursive Perceptual Representations	NIPS 2012
79.6 %	An Analysis of Single-Layer Networks in Unsupervised Feature Learning	AISTATS 2011

[2] rodrigob.github.io

preprocessing[3]

The new model I used benefit from two preprocessing techniques:

1. PCA whitening
2. Kmeans
3. plug in our two-layer neural network (the original paper use SVM at the end)

[3] Adam Coates, Andrew Y Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning". In: *International conference on artificial intelligence and statistics*. 2011, pp. 215–223.

high level description

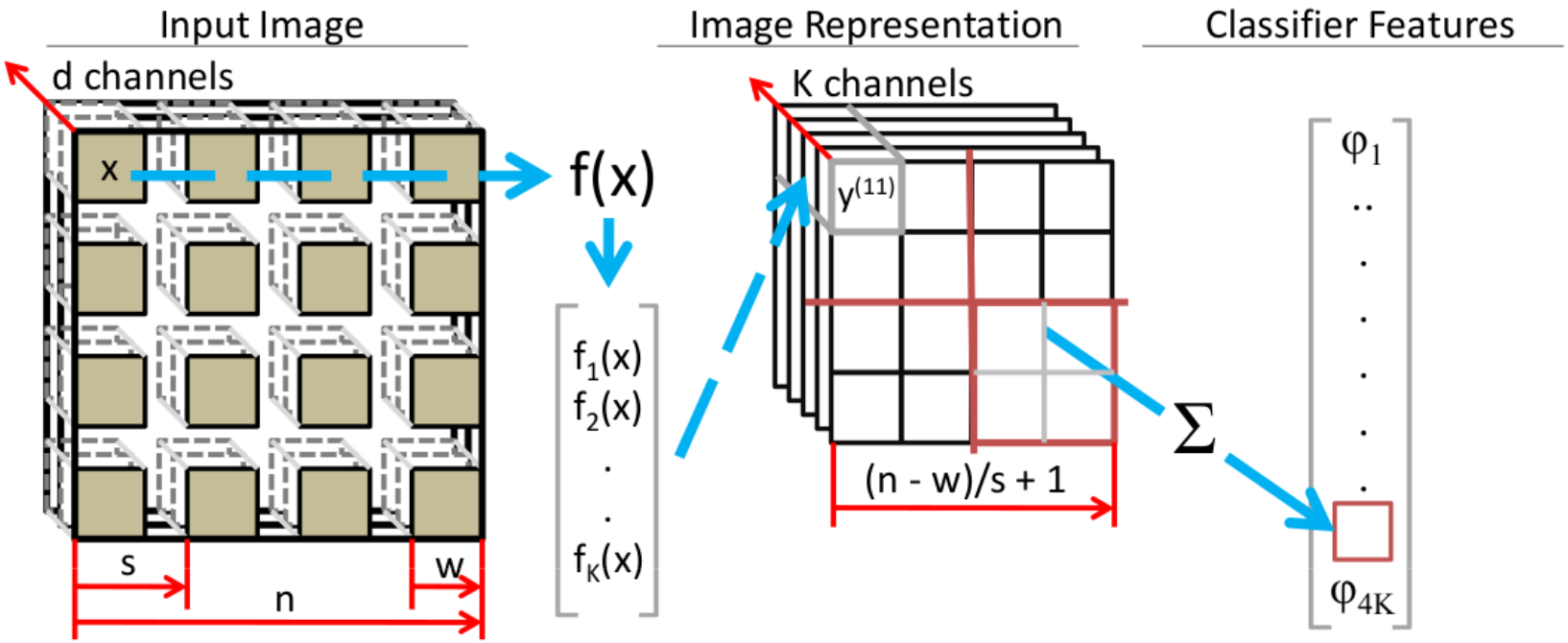
Learn a feature representation:

1. Extract random patches from unlabeled training images.
2. Apply a pre-processing stage to the patches.
3. Learn a feature-mapping using an unsupervised learning algorithm.

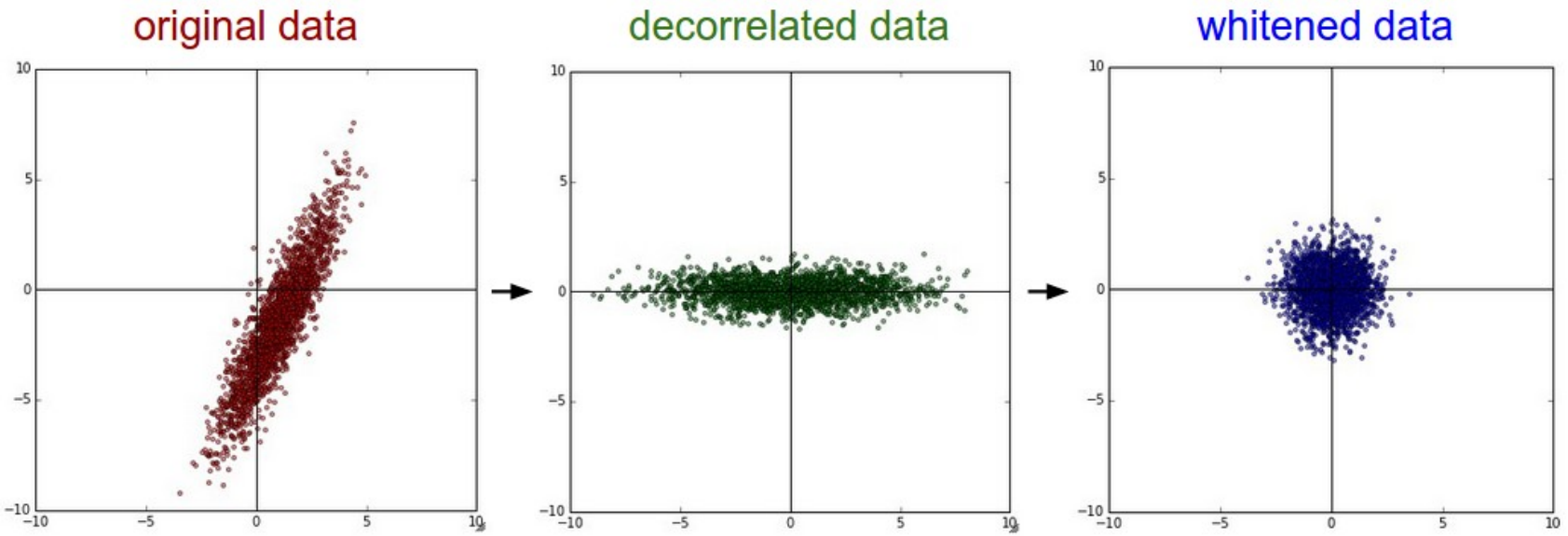
Given the learned feature mapping, we can then perform feature extraction:

1. Break an image into patches.
2. Cluster these patches.
3. Concatenate cluster result of each patch $\{0,0,\ldots,1,\ldots,0\}$, as new representation of this image.

steps

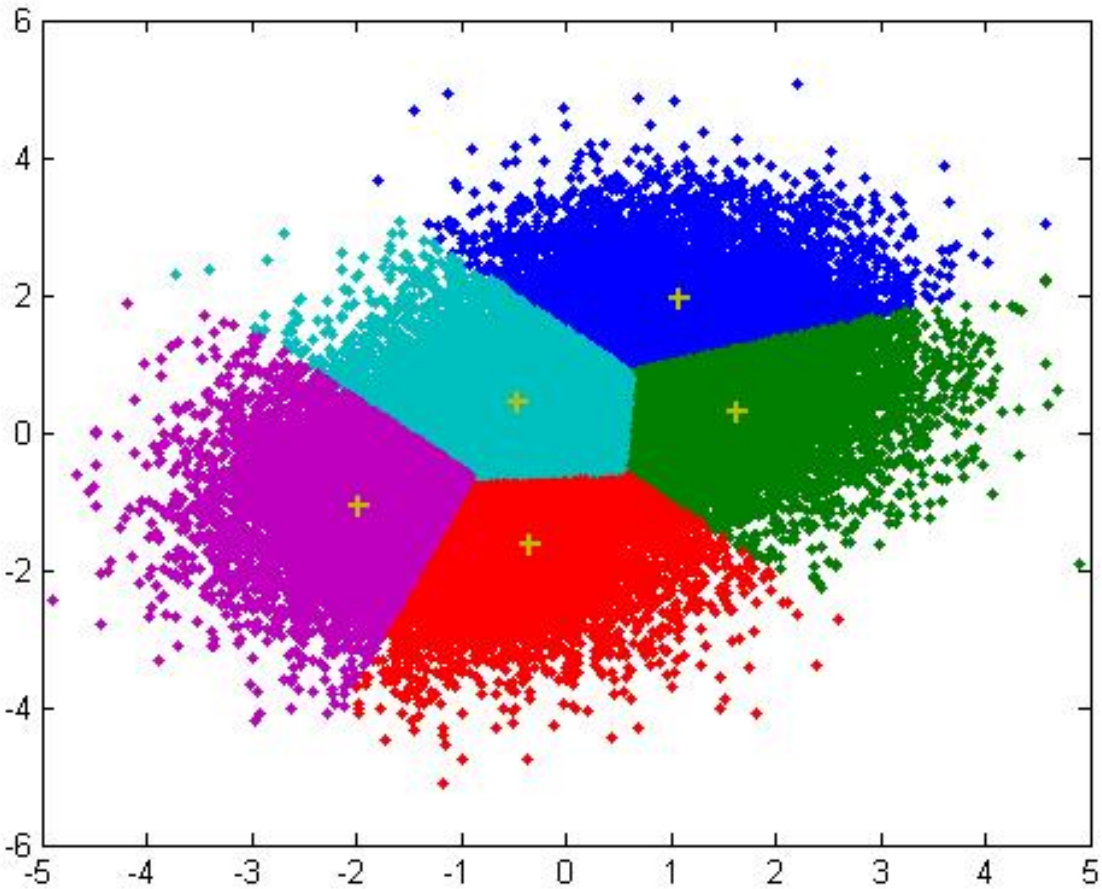

Input Image
d channels
x
s
n
w
f(x)
$f_1(x)$
$f_2(x)$
$f_K(x)$
Image Representation
K channels
$y^{(11)}$
(n - w)/s + 1
Σ
Classifier Features
φ_1
φ_{4K}

PCAwhitening visualize



Use PCAwhitening without dimention reduction.

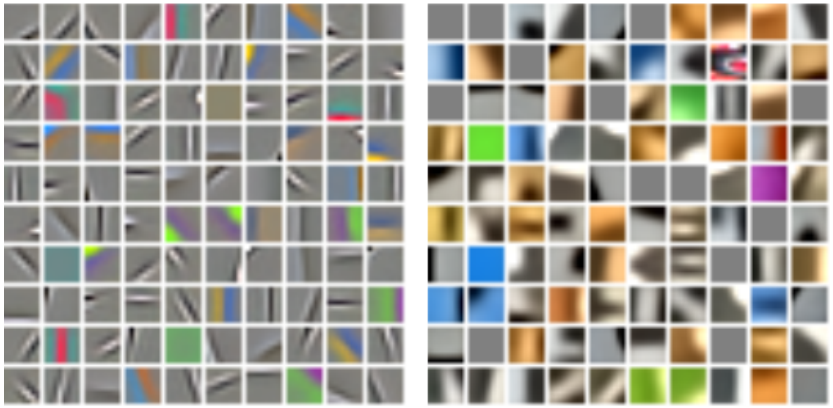
Kmeans visualize



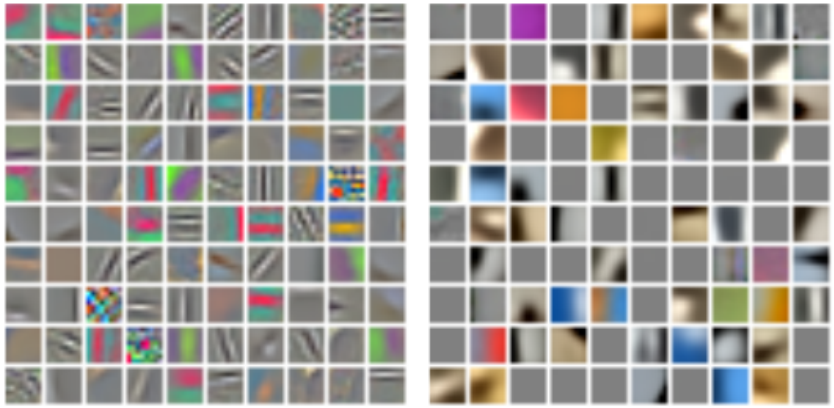
Select 1600 clusters

PCAwhitening effect on Kmeans

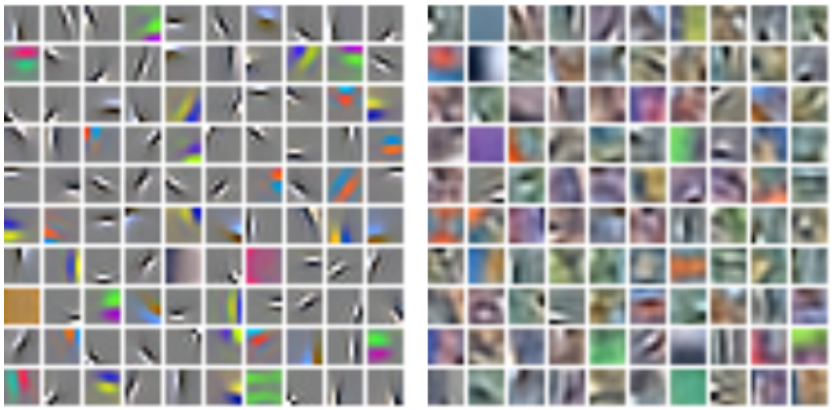
Some cluster centroids



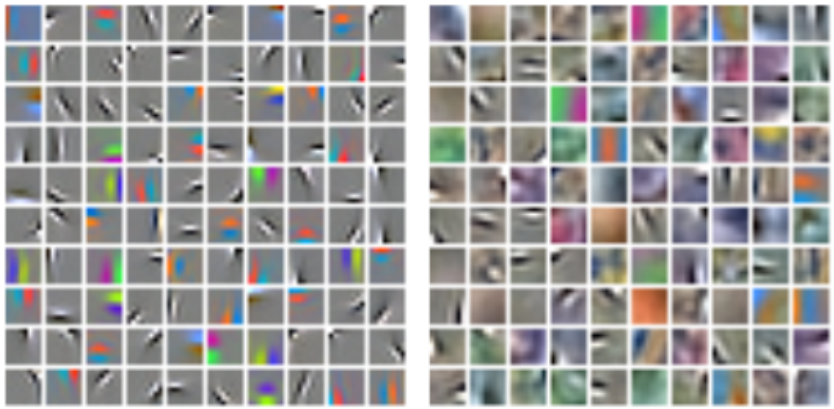
(a) K-means (with and without whitening)



(b) GMM (with and without whitening)

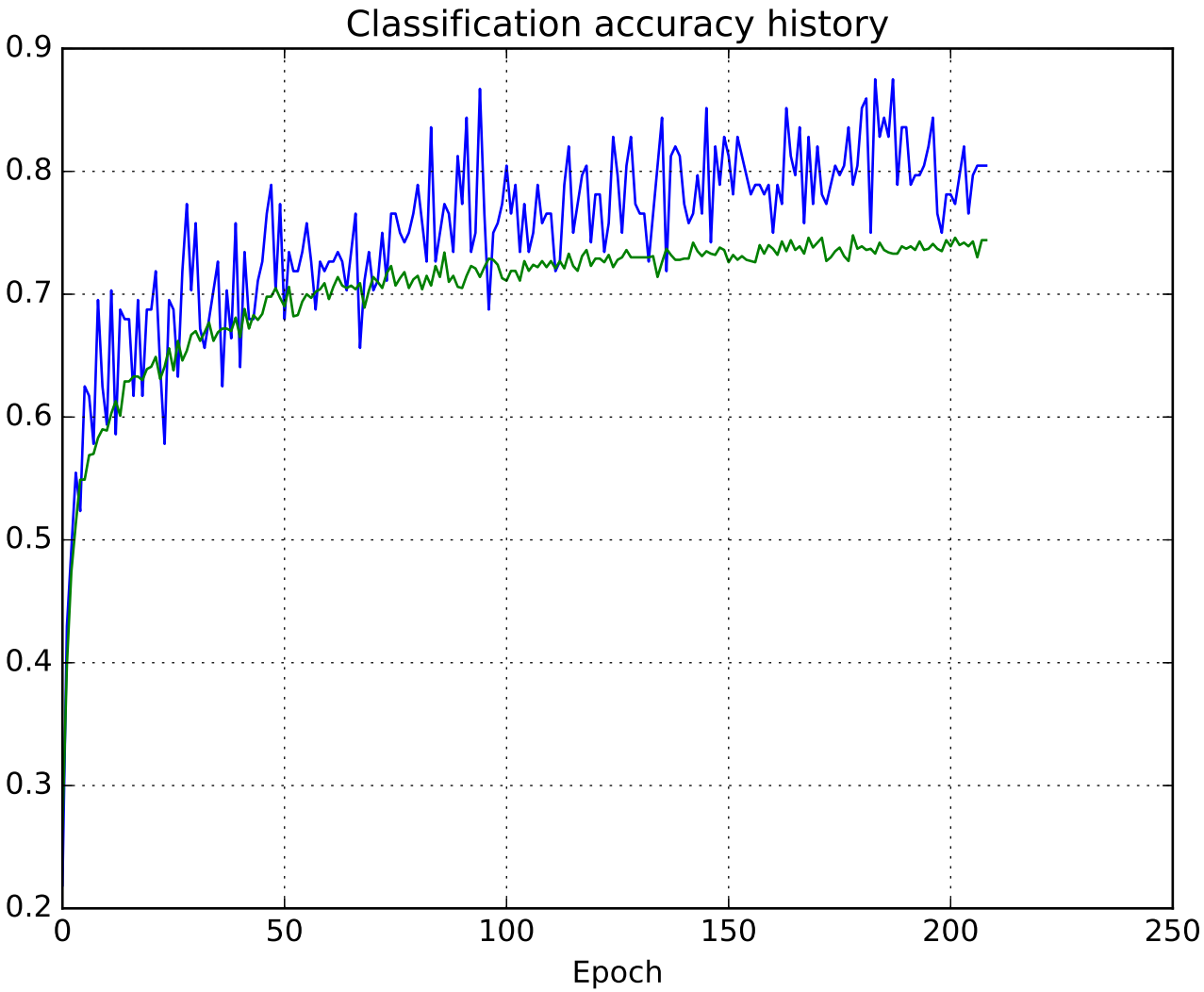


(c) Sparse Autoencoder (with and without whitening)



(d) Sparse RBM (with and without whitening)

When should we stop training?



more information from results

	Naive	Dropout	Preprocessed
hidden nodes	350	500	200
learning rate	1×10^{-3}	1×10^{-4}	5×10^{-4}
learning rate Decay	.95	.95	.99
regularization	L2,0.05	Dropout,.5	Dropout,.3
Activation	ReLU	Leaky ReLU	ReLU
Update method	SGD	Momentum,0.9	Momentum,0.95
Iterations	1×10^4	1×10^4	7×10^4
Batch size	100	100	128
Time(min)	15	80	110
Train accuracy	60%	65%	80%
Validation	55%	62%	75%
Test	52%	55%	74%

importance of mean image substraction

The result I got is 75%, the orignal paper get 79%.
It's because I forgot to subtract mean before doing PCA whitening.
After fix this bug, accuracy increases to 77%. Much closer.

Huge difference! Mean image substraction is important.

questions on PCAwhitening and Kmeans?

1. mp1
   - tricks
   - new model
2. mp2
   - tricks
   - choosing from different models
   - delving into one model

Goal

- Input: CIFAR 100 image
- Archtecture: Not determined
- Output:prediction among 20 classes

tricks that show little difference in my experiments

- Dropout
- Update methods
- PCA whitening and Kmeans

Initialization methods

Becomes more and more important when network goes deep.
Recall that we have two problems: gradient vanishing $(\beta_w \beta_\alpha)^p \ll 1$ and gradient exploding $(\beta_w \beta_\alpha)^p \gg 1$:

- Orthogonal initalization
- LUSV initalization
- Xavier initialization
- Kaiming He[4] initialization method(works best)

[4] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1026–1034.

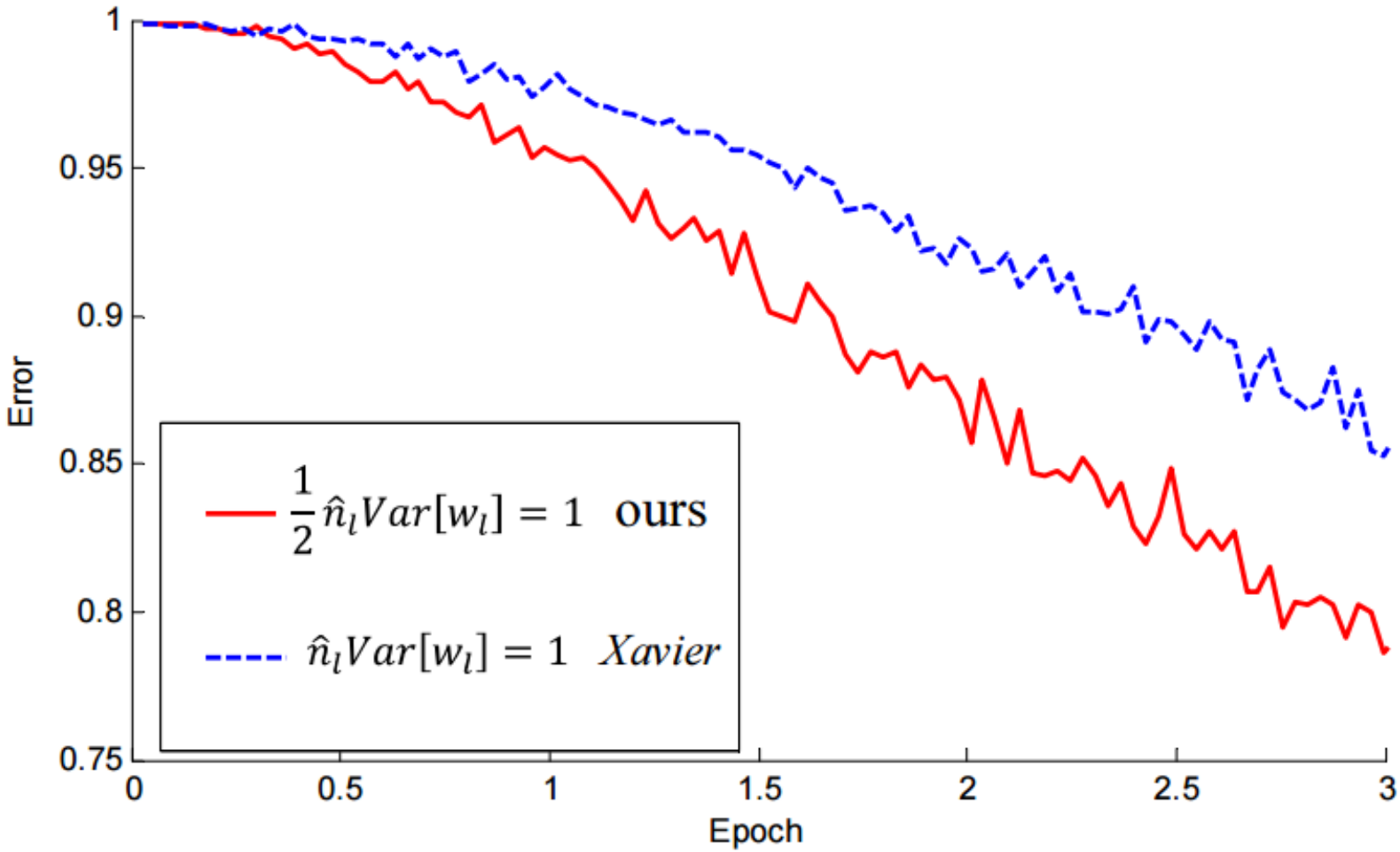
Kaiming He's initialization method

The idea is scale backward pass signal to 1 at each layer.
Implementation is very simple.

$$std = sqrt(2/Depth_{in}/receptionFieldSize).$$

$Depth_{in}$: number of filters of previous layer comes in.
receptionFieldSize: eg. 3x3

could to make 30 layers deep net converge



number of hidden neurons

- More hidden neurons may not show any superior, only increasing time cost.
- Adding hidden layers sometimes make things worse. Kaiming He[5] found that about 30% redundant computation comes from the fully connected layers. Fully connected layer is less efficient than conv layer.

One solution: replace the fully connected layer between the last conv layer and hidden layer with global average pooling.

[5] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385* (2015).

New model

How do we improve it?

To my knowledge, I found these possible way to improve accuracy:

- XNOR net[6]
- mimic learning[7] (model compression)
- switch to faster framework(mxnet[8]), rather than tensorflow :)
- residual neural network[9]

---

[6] Mohammad Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *arXiv preprint arXiv:1603.05279* (2016).

[7] Jimmy Ba and Rich Caruana. "Do deep nets really need to be deep?" In: *Advances in neural information processing systems*. 2014, pp. 2654–2662.

[8] Tianqi Chen et al. "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems". In: *arXiv preprint arXiv:1512.01274* (2015).

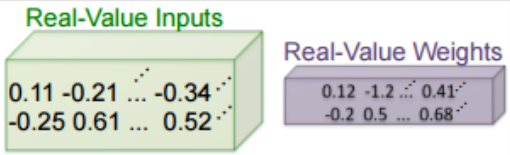
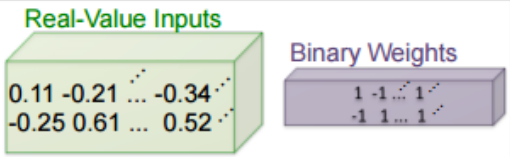
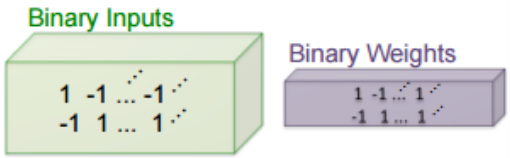
[9] He et al., "Deep Residual Learning for Image Recognition".

what is XNOR net?

$$x^b = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p, \end{cases}$$

$\sigma(x)$, activation function

XNOR net speed

	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Time Saving on CPU (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution		+ , − , ×	1x	1x	%56.7
• Binary Weight		+ , −	~32x	~2x	%53.8
BinaryWeight Binary Input **(XNOR-Net)**		XNOR , bitcount	~32x	~58x	%44.2

what is mimic learning, basic idea

With a high accuracy teacher model,
we not only tell the student neural network which label is true or wrong(0,1)
also tell the student neural network some classes are close to each other and some are not.

Example

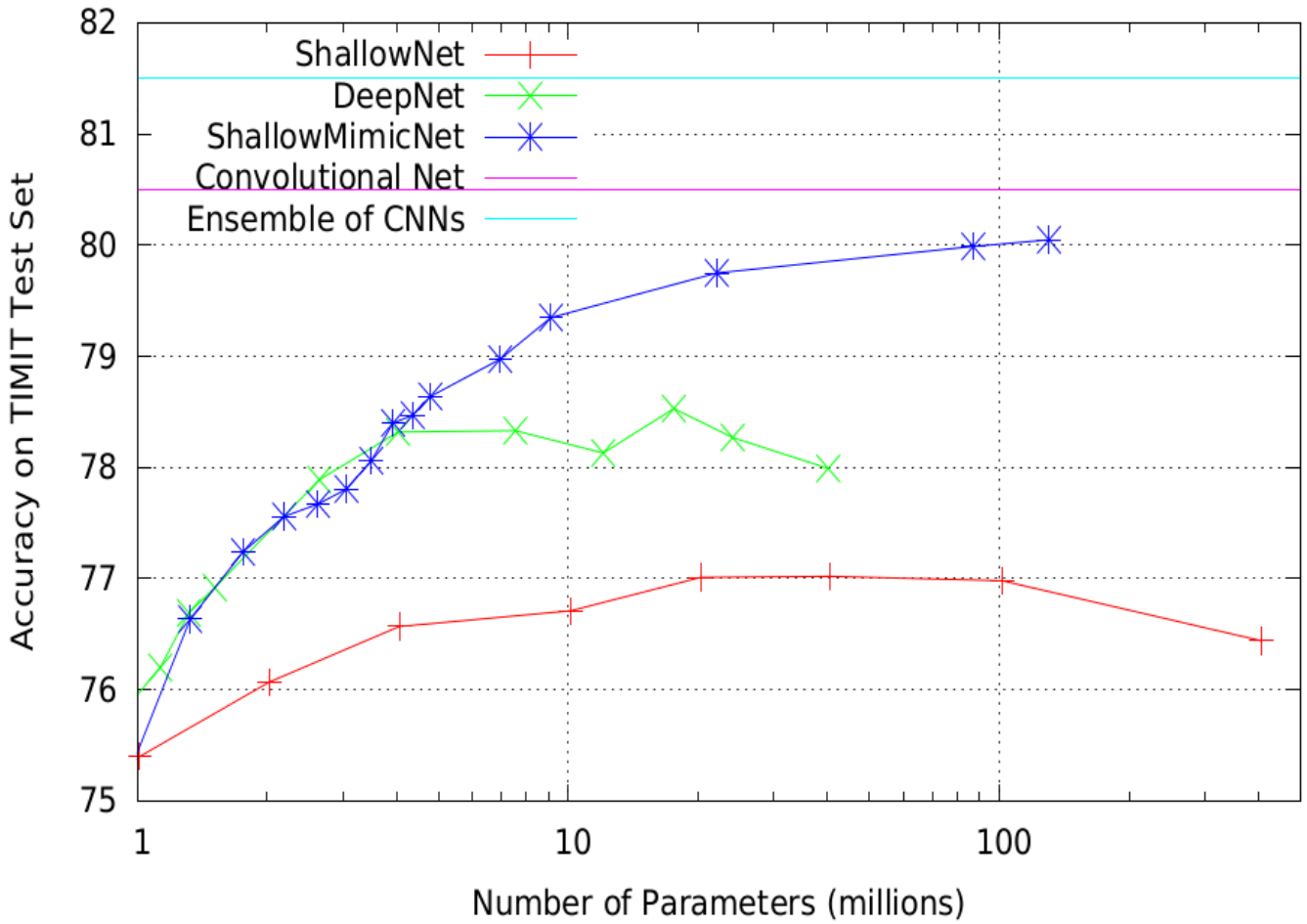
In CIFAR10, truck and car are in different classes, however, they share some common features. So when there's a car in the image, truck's probability is also high. Teacher model helps student model jointly learn these two concepts.

what is mimic learning, details

High level overview:

1. train a state-of-the-art neural network
2. get the $log(p_{deep}(y|X))$ for training set
3. replace the softmax layer of shallow neural network with a linear regressor
4. minimize log probability error:
   $J(\theta) = \sum_{y \in labels}(log(p(y|X)) - log(p_{deep}(y|X)))^2$
5. put back softmax layer
6. fine tuning

result from paper



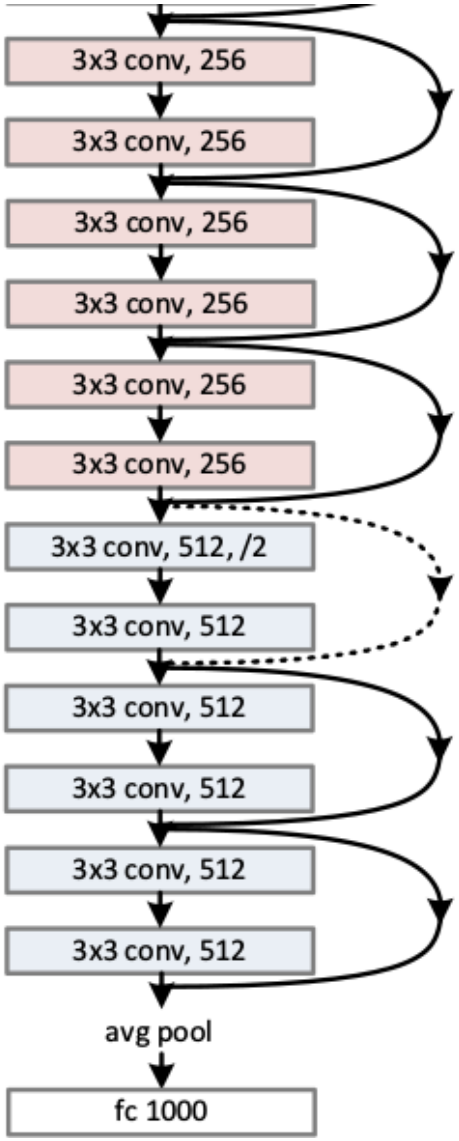
residual neural network

Basic idea:
Learn $f(x) - x$ instead of $f(x)$.

residual neural network

The only two differences between residual neural network and ConvNet:

1. no hidden layers
2. use shortcut module, which allows a layer skip the layer on top of it, and pass its value to the next layer.



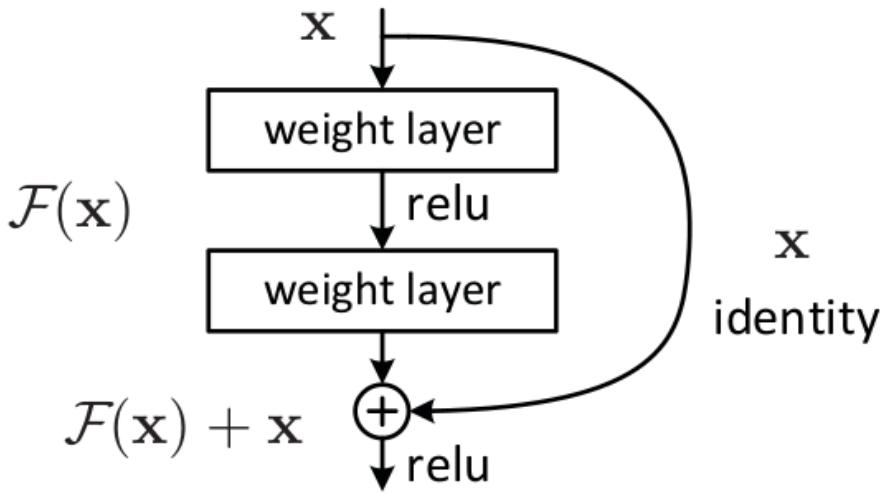
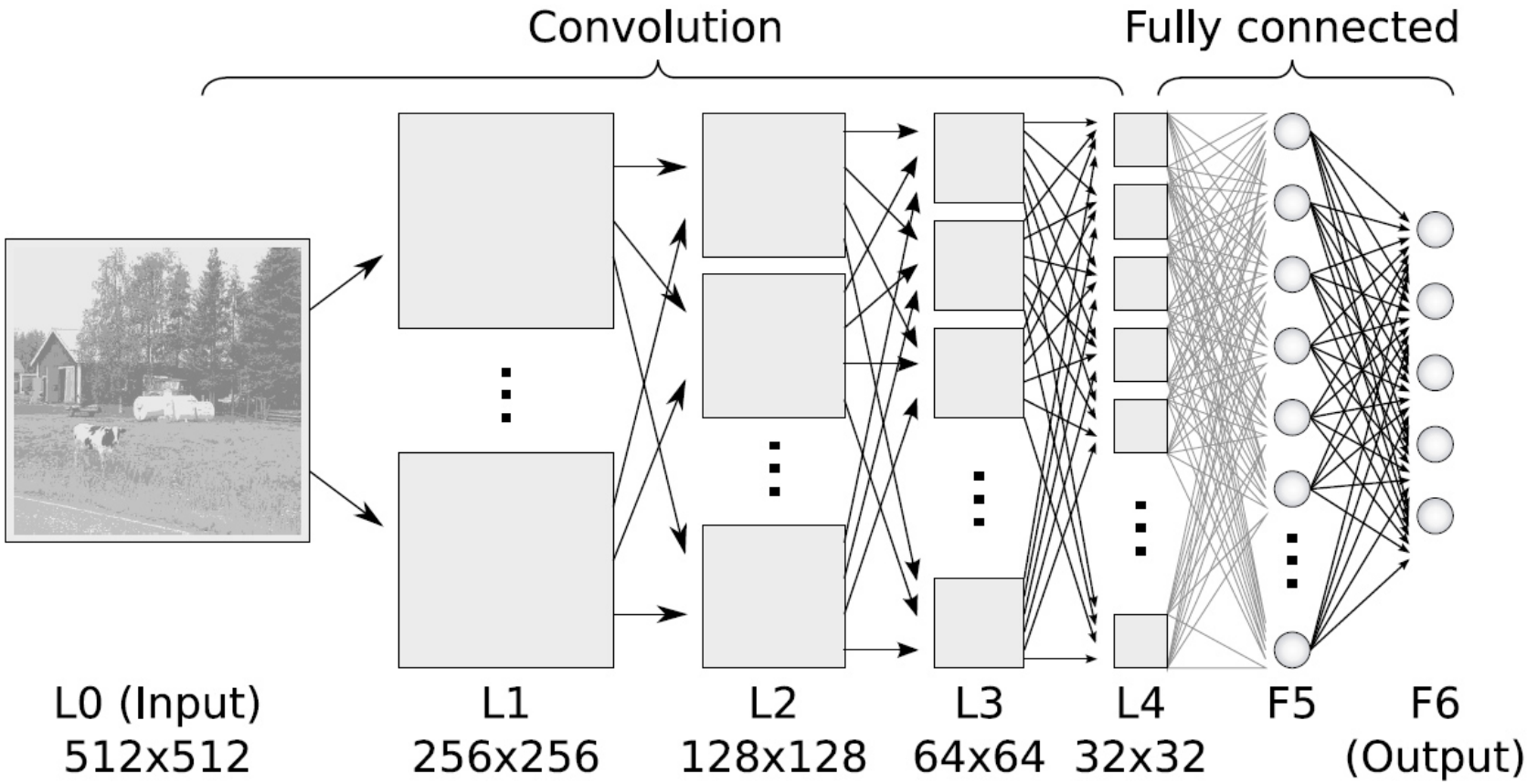


Figure 2. Residual learning: a building block.

traditional convolutional neural network



parameters on each layers

A commonly used VGGnet:

```
conv3-64  x 2 :  38,720
conv3-128 x 2 :  221,440
conv3-256 x 3 :  1,475,328
conv3-512 x 3 :  5,899,776
conv3-512 x 3 :  7,079,424
fc1           :  102,764,544
fc2           :  16,781,312
fc3           :  4,097,000
TOTAL         :  138,357,544
```

Notice that 74% parameters are from fc1, however, actual accuracy improvement is from conv layers. Residual nerual network, instead, uses all convolution layers and a global average pooling layer at the end.

Jost Tobias Springenberg et al. "Striving for simplicity: The all convolutional net". In: *arXiv preprint arXiv:1412.6806* (2014),

architecture comparison

Table: Differences between three archectures

	AlexNet	Kmeans	ResNet
parameters	1M	.4M	.13M
Layers	7	3	14
learning rate	.1	5×10^{-4}	.1
regularization	L2	Dropout,.3	None
epoch	10	140	18
Batch size	128	128	256
Time(min)	180	80	180
CIFAR10 Acc	82%	75%	84%
Train accuracy	90%	80%	86%
Test	56%	56%	63%

why residual neural network more efficient?

1. Less trainable parameters than neural networks that have the same depth.
2. Lower layer response.
3. shortcut module allows error δ directly pass to previous layers, instead of going through each layer. It implicitly makes a deeper network shallower, so it won't suffer much from gradient vanishing and exploding. It makes training faster.

Code, report and papers can be access via github:
mp1
https://github.com/yihui-he/Single-Layer-neural-network-with-PCAwhitening-Kmeans
mp2
https://github.com/yihui-he/Residual-neural-network

questions?