# Image Recognition

Wei-Ta Chu

# AlexNet

Wei-Ta Chu

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” NIPS, 2012.

# Introduction



- To learn about thousands of objects from millions of images, we need a model with a large learning capacity.
- CNNs' capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images.
- CNNs have still been prohibitively expensive to apply in large scale to high-resolution images. Current GPUs, paired with a highly-optimized implementation of 2D convolution, are powerful enough to facilitate the training of CNNs.

# Introduction



- Contributions
  - We trained one of the largest convolutional neural networks to date on the subsets of ImageNet and achieved by far the best results
  - Our network contains a number of new and unusual features which improve its performance and reduce its training time.
  - We used several effective techniques for preventing overfitting
- Our network takes between five and six days to train on two GTX 580 3GB GPUs.

# Dataset



- ImageNet: Over 15 million labeled high-resolution images belonging to roughly 22,000 categories.
- Labeled by human labelers using Amazon's Mechanical Turk crowd-sourcing tool.
- Starting from 2010, ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held.
- ILSVRC uses a subset of ImageNet with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images.

# Dataset



- ILSVRC-2010 is the only version of ILSVRC for which the test set labels are available, on which we performed most of our experiments.
- Down-sampled the images to a fixed resolution of 256 × 256. Given a rectangular image, we first rescaled the image such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image.
- Subtracting the mean activity over the training set from each pixel. So we trained our network on the (centered) raw RGB values of the pixels.

# Architecture



- ReLU Nonlinearity
- The standard way to model a neuron's output $f$ as a function of its input $x$ is with $f(x) = tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$
- In terms of training time with gradient descent, these saturating nonlinearities are much slower than $f(x) = max(0,x)$
- Faster learning has a great influence on the performance of large models trained on large datasets.

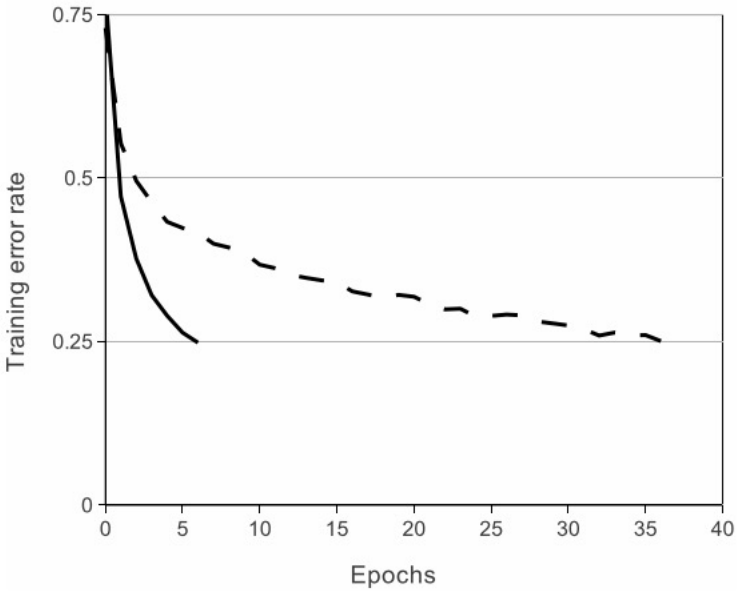



Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (**dashed line**). The learning rates for each net-

# Architecture



- Training on Multiple GPUs
- We spread the net across two GPUs. The parallelization scheme essentially puts half of the kernels (or neurons) on each GPU, with one additional trick: the GPUs communicate only in certain layers.
- The kernels of layer 3 take input from all kernel maps in layer 2. However, kernels in layer 4 take input only from those kernel maps in layer 3 which reside on the same GPU.

# Architecture



- Local Response Normalization
- Denote $a^i_{x,y}$ the activity of a neuron computed by applying kernel $i$ at position $(x, y)$ and then applying the ReLU nonlinearity. The response-normalized

$$b^i_{x,y} = a^i_{x,y} / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

where the sum runs over $n$ "adjacent" kernel maps at the same spatial position, and $N$ is the total number of kernels in the layer.

# Architecture



- Local Response Normalization
- This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels.

# Architecture



- Overlapping Pooling
- A pooling layer can be thought of as consisting of a grid of pooling units spaced $s$ pixels apart, each summarizing a neighborhood of size $z \times z$ centered at the location of the pooling unit. If we set $s = z$, we obtain traditional local pooling as commonly employed in CNNs.
- If we set $s < z$, we obtain overlapping pooling. This is what we use throughout our network, with $s = 2$ and $z = 3$.

# Architecture



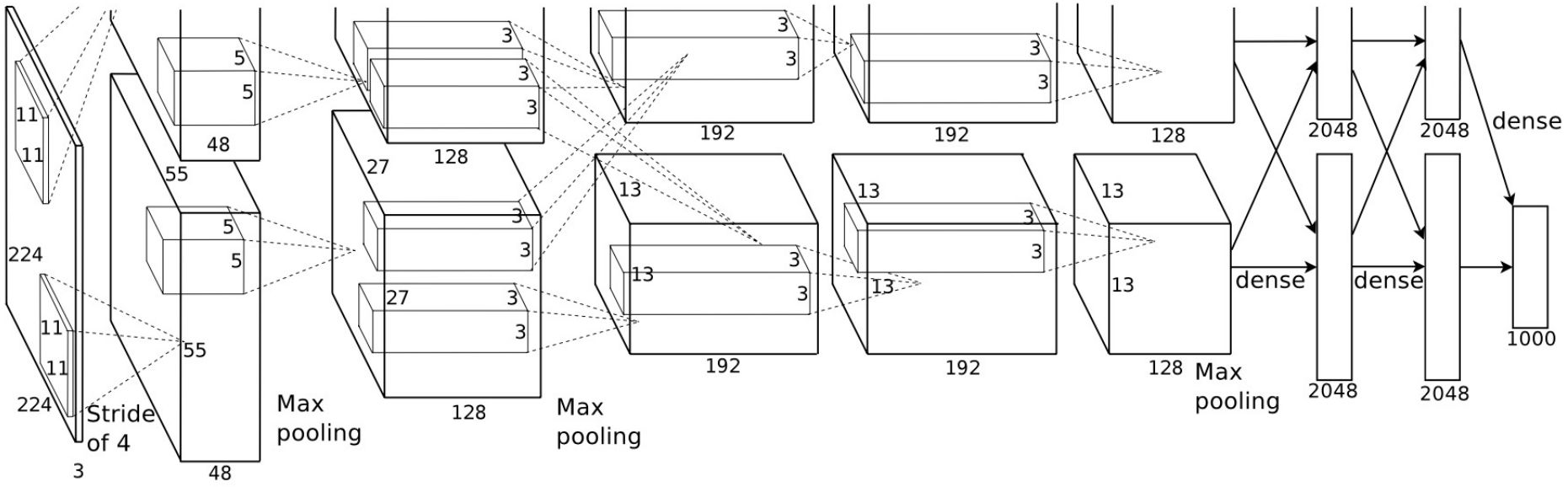



Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

# Architecture



- Overall architecture
- The net contains eight layers with weights; the first five are convolutional and the remaining three are fully-connected.
- The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels.
- Our network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction distribution.
- The network architecture has 60 million parameters.

# Architecture



- The kernels of the second, fourth, and fifth convolutional layers are connected only to those kernel maps in the previous layer which reside on the same GPU.
- Response-normalization layers follow the first and second convolutional layers. Max-pooling layers follow both response-normalization layers as well as the fifth convolutional layer. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

# Architecture



- The first convolutional layer filters the 224 × 224 × 3 input image with 96 kernels of size 11 × 11 × 3 with a stride of 4 pixels.
- The second convolutional layer filters input with 256 kernels of size 5 × 5 × 48.
- The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size 3 × 3 × 256.
- The fourth convolutional layer has 384 kernels of size 3 × 3 × 192 , and the fifth convolutional layer has 256 kernels of size 3 × 3 × 192.
- The fully-connected layers have 4096 neurons each.

# Reducing Overfitting



- Data Augmentation
- Generating image translations and horizontal reflections
- We do this by extracting random 224 × 224 patches (and their horizontal reflections) from the 256×256 images and training our network on these extracted patches.
- At test time, the network makes a prediction by extracting five 224 × 224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

# Reducing Overfitting



- Data Augmentation
- Altering the intensities of the RGB channels in training images.
- We add multiples of the found principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1.
- To each RGB image pixel $I_{xy} = [I_{xy}^R , I_{xy}^G , I_{xy}^B ]^T$ we add the following quantity:

$$[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

where $\mathbf{p}_i$ and $\lambda_i$ are $i$th eigenvector and eigenvalue of the $3 \times 3$ covariance matrix of RGB pixel values.

# Reducing Overfitting



- Dropout
- Setting to zero the output of each hidden neuron with probability 0.5. The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in back- propagation.
- Every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons.

# Details of learning



- Train the model using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005.
- We initialized the weights in each layer from a zero-mean Gaussian distribution with standard deviation 0.01. We initialized the neuron biases in the second, fourth, and fifth convolutional layers, as well as in the fully-connected hidden layers, with the constant 1.
- Divide the learning rate by 10 when the validation error rate stopped improving with the current learning rate.
- We trained the network for roughly 90 cycles through the training set of 1.2 million images, which took five to six days on two NVIDIA GTX 580 3GB GPUs.

# Results



| Model | Top-1 | Top-5 |
|---|---|---|
| *Sparse coding [2]* | *47.1%* | *28.2%* |
| *SIFT + FVs [24]* | *45.7%* | *25.7%* |
| CNN | **37.5%** | **17.0%** |

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

| Model | Top-1 (val) | Top-5 (val) | Top-5 (test) |
|---|---|---|---|
| *SIFT + FVs [7]* | — | — | *26.2%* |
| 1 CNN | 40.7% | 18.2% | — |
| 5 CNNs | 38.1% | 16.4% | **16.4%** |
| 1 CNN* | 39.0% | 16.6% | — |
| 7 CNNs* | 36.7% | 15.4% | **15.3%** |

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk* were "pre-trained" to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

# Results



- The convolutional kernels learned by the network's two data-connected layers.

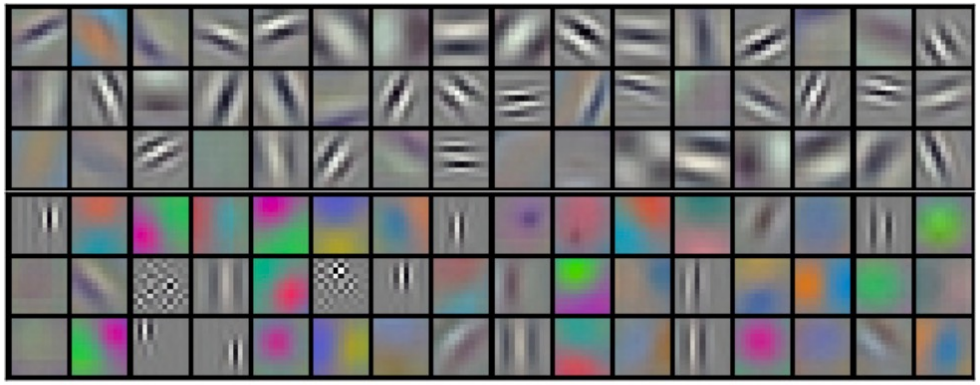


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

# Results



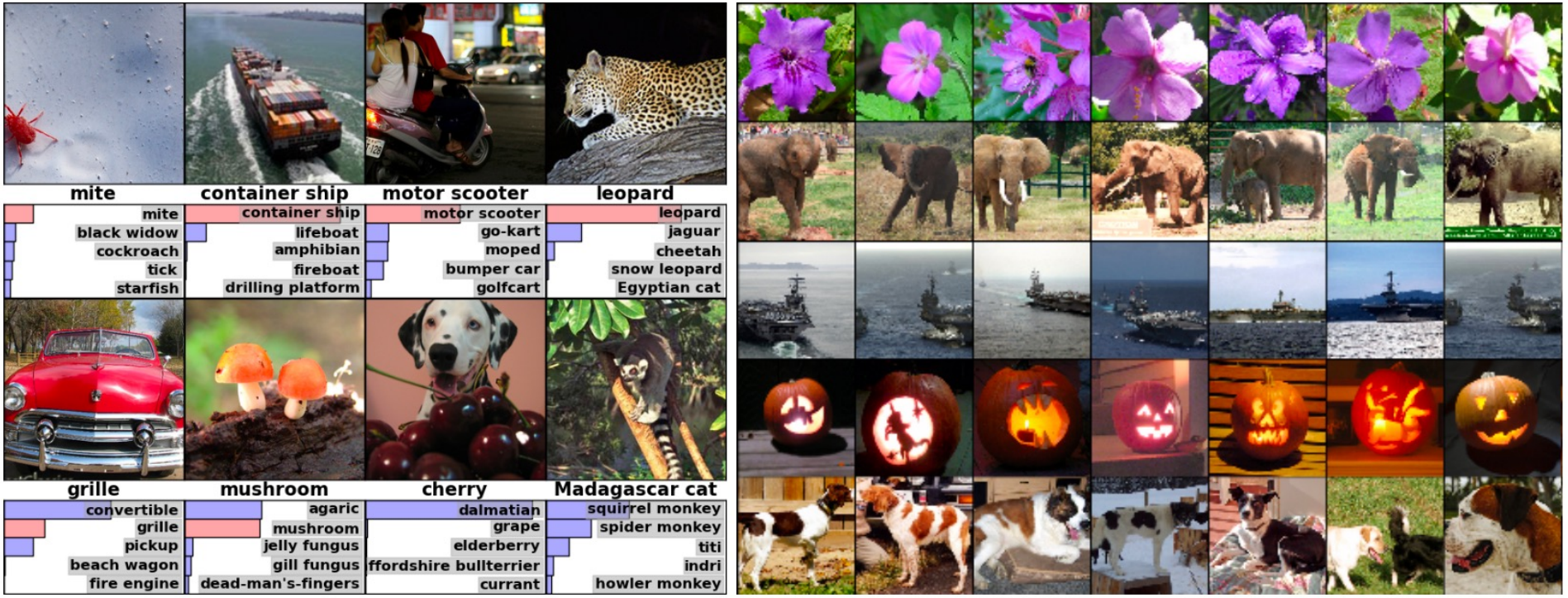



Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

# Discussion



- Our results show that a large, deep convolutional neural network is capable of achieving record- breaking results on a highly challenging dataset using purely supervised learning.
- Removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So the depth really is important for achieving our results.

# VGG

Wei-Ta Chu

Karen Simonyan and Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” ICLR, 2015.

# Introduction



- We investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting.
- Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small ($3 \times 3$) convolution filters.

# ConvNet Configurations



- The input is a fixed-size 224 × 224 RGB image. The only pre-processing is subtracting the mean RGB value, computed on the training set, from each pixel.
- The image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3 × 3.
- The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution.

# ConvNet Configurations



- Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers.
- Max-pooling is performed over a $2 \times 2$ pixel window, with stride 2.
- A stack of convolutional layers is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification.
- All hidden layers are equipped with ReLU non-linearity. No Local Response Normalisation.

# ConvNet Configurations



Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

# ConvNet Configurations



- Discussion
  - Rather than using relatively large receptive fields in the first conv. layers, or 7 × 7 with stride 2, we use very small 3 × 3 receptive fields throughout the whole net.
  - A stack of two 3 × 3 conv. layers (without spatial pooling in between) has an effective receptive field of 5 × 5; three such layers have a 7 × 7 effective receptive field.
  - So what have we gained by using, for instance, a stack of three 3 × 3 conv. layers instead of a single 7 × 7 layer?

# ConvNet Configurations



- Discussion
  - First, we incorporate three non-linear rectification layers instead of a single one, which makes the decision function more discriminative.
  - Second, we decrease the number of parameters: assuming that both the input and the output of a three-layer $3 \times 3$ convolution stack has $C$ channels, the stack is parametrised by $3\ (3^2C^2) = 27C^2$ weights; at the same time, a single $7 \times 7$ conv. layer would require $7^2C^2 = 49C^2$ parameters, i.e. 81% more. This can be seen as imposing a regularisation on the $7 \times 7$ conv. filters.

# Classification Framework -- Training



- The training is carried out by optimising the multinomial logistic regression objective using mini-batch gradient descent with momentum.
- Dropout regularisation for the first two fully-connected layers.
- The learning rate was initially set to $10^{-2}$, and then decreased by a factor of 10 when the validation set accuracy stopped improving.

# Classification Framework -- Training



- The initialisation of the network weights is important.
- We began with training the configuration A (Table 1), shallow enough to be trained with random initialisation.
- Then, when training deeper architectures, we initialised the first four convolutional layers and the last three fully-connected layers with the layers of net A (the intermediate layers were initialised randomly).

# Classification Framework -- Training



- To obtain the fixed-size 224×224 ConvNet input images, they were randomly cropped from rescaled training images (one crop per image per SGD iteration).
- To further augment the training set, the crops underwent random horizontal flipping and random RGB colour shift

# Classification Framework -- Testing



- An input image is isotropically rescaled to a pre-defined smallest image side, denoted as $Q$. We note that $Q$ is not necessarily equal to the training scale $S$.
- The fully-connected layers are first converted to convolutional layers (the first FC layer to a $7 \times 7$ conv. layer, the last two FC layers to $1 \times 1$ conv. layers). The resulting fully-convolutional net is then applied to the whole (uncropped) image. The result is a class score map with the number of channels equal to the number of classes.

# Classification Framework -- Testing



- Finally, to obtain a fixed-size vector of class scores for the image, the class score map is spatially averaged (sum-pooled).
- We also augment the test set by horizontal flipping of the images; the soft-max class posteriors of the original and flipped images are averaged to obtain the final scores for the image.

# Classification Framework – Implementation Details



- Multi-GPU training exploits data parallelism, and is carried out by splitting each batch of training images into several GPU batches, processed in parallel on each GPU.
- After the GPU batch gradients are computed, they are averaged to obtain the gradient of the full batch.
- On a system equipped with four NVIDIA Titan Black GPUs, training a single net took 2–3 weeks depending on the architecture

# Classification Experiments



- ILSVRC-2012 dataset
- The dataset includes images of 1000 classes, and is split into three sets: training (1.3M images), validation (50K images), and testing (100K images with held-out class labels).
- For the majority of experiments, we used the validation set as the test set.

# Classification Experiments



- Single scale evaluation
- The test image size was set as follows: $Q = S$ for fixed $S$, and $Q = 0.5(S_{min} + S_{max})$ for jittered $S$.

Table 3: **ConvNet performance at a single test scale.**

| ConvNet config. (Table 1) | smallest image side | | top-1 val. error (%) | top-5 val. error (%) |
|---|---|---|---|---|
| | train ($S$) | test ($Q$) | | |
| A | 256 | 256 | 29.6 | 10.4 |
| A-LRN | 256 | 256 | 29.7 | 10.5 |
| B | 256 | 256 | 28.7 | 9.9 |
| C | 256 | 256 | 28.1 | 9.4 |
| | 384 | 384 | 28.1 | 9.3 |
| | [256;512] | 384 | 27.3 | 8.8 |
| D | 256 | 256 | 27.0 | 8.8 |
| | 384 | 384 | 26.8 | 8.7 |
| | [256;512] | 384 | 25.6 | 8.1 |
| E | 256 | 256 | 27.3 | 9.0 |
| | 384 | 384 | 26.9 | 8.7 |
| | [256;512] | 384 | **25.5** | **8.0** |

# Classification Experiments



- Using local response normalisation (A-LRN network) does not improve on the model A without any normalisation layers.
- Classification error decreases with the increased ConvNet depth: from 11 layers in A to 19 layers in E.

# Classification Experiments



- The configuration C (which contains three 1 × 1 conv. layers), performs worse than the configuration D, which uses 3 × 3 conv. layers throughout the network. While the additional non-linearity does help (C is better than B), it is also important to capture spatial context by using conv. filters with non-trivial receptive fields (D is better than C).
- The error rate of our architecture saturates when the depth reaches 19 layers, but even deeper models might be beneficial for larger datasets.

# Classification Experiments



- Scale jittering at training time ($S \in [256; 512]$) leads to significantly better results than training on images with fixed smallest side ($S = 256$ or $S = 384$), even though a single scale is used at test time.
- This confirms that training set augmentation by scale jittering is indeed helpful for capturing multi-scale image statistics.

# Classification Experiments



- Multi-scale evaluation
- Running a model over several rescaled versions of a test image (corresponding to different values of $Q$), followed by averaging the resulting class posteriors.

# Classification Experiments



- The models trained with fixed $S$ were evaluated over three test image sizes, close to the training one: $Q = \{S - 32, S, S + 32\}$.
- The model trained with variable $S \in [S_{min}; S_{max}]$ was evaluated over a larger range of sizes $Q = \{S_{min}, 0.5(S_{min} + S_{max}), S_{max}\}$.

# Classification Experiments



- Scale jittering at test time leads to better performance
- The deepest configurations (D and E) perform the best

Table 4: **ConvNet performance at multiple test scales.**

| ConvNet config. (Table 1) | smallest image side | | top-1 val. error (%) | top-5 val. error (%) |
|---|---|---|---|---|
| | train ($S$) | test ($Q$) | | |
| B | 256 | 224,256,288 | 28.2 | 9.6 |
| C | 256 | 224,256,288 | 27.7 | 9.2 |
| | 384 | 352,384,416 | 27.8 | 9.2 |
| | [256; 512] | 256,384,512 | 26.3 | 8.2 |
| D | 256 | 224,256,288 | 26.6 | 8.6 |
| | 384 | 352,384,416 | 26.5 | 8.6 |
| | [256; 512] | 256,384,512 | **24.8** | **7.5** |
| E | 256 | 224,256,288 | 26.9 | 8.7 |
| | 384 | 352,384,416 | 26.7 | 8.6 |
| | [256; 512] | 256,384,512 | **24.8** | **7.5** |

# Classification Experiments



- Comparison with the state of the arts

Table 7: **Comparison with the state of the art in ILSVRC classification**. Our method is denoted as "VGG". Only the results obtained without outside training data are reported.

| Method | top-1 val. error (%) | top-5 val. error (%) | top-5 test error (%) |
|---|---|---|---|
| VGG (2 nets, multi-crop & dense eval.) | **23.7** | **6.8** | **6.8** |
| VGG (1 net, multi-crop & dense eval.) | 24.4 | 7.1 | 7.0 |
| VGG (ILSVRC submission, 7 nets, dense eval.) | 24.7 | 7.5 | 7.3 |
| GoogLeNet (Szegedy et al., 2014) (1 net) | - | 7.9 | |
| GoogLeNet (Szegedy et al., 2014) (7 nets) | - | **6.7** | |
| MSRA (He et al., 2014) (11 nets) | - | - | 8.1 |
| MSRA (He et al., 2014) (1 net) | 27.9 | 9.1 | 9.1 |
| Clarifai (Russakovsky et al., 2014) (multiple nets) | - | - | 11.7 |
| Clarifai (Russakovsky et al., 2014) (1 net) | - | - | 12.5 |
| Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets) | 36.0 | 14.7 | 14.8 |
| Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net) | 37.5 | 16.0 | 16.1 |
| OverFeat (Sermanet et al., 2014) (7 nets) | 34.0 | 13.2 | 13.6 |
| OverFeat (Sermanet et al., 2014) (1 net) | 35.7 | 14.2 | - |
| Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets) | 38.1 | 16.4 | 16.4 |
| Krizhevsky et al. (Krizhevsky et al., 2012) (1 net) | 40.7 | 18.2 | - |

# Conclusion



- We evaluated very deep convolutional networks (up to 19 weight layers) for large-scale image classification.
- The representation depth is beneficial for the classification accuracy

# ResNet

Wei-Ta Chu

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep Residual Learning for Image Recognition,” CVPR, 2016.

# Introduction



- Recent evidence reveals that network depth is of crucial importance.
- A question arises: Is learning better networks as easy as stacking more layers?
  - vanishing/exploding gradient problem
  - addressed by normalized initialization and intermediate normalization layers
- With the network depth increasing, accuracy gets saturated and then degrades rapidly.
  - such degradation is not caused by overfitting

# Introduction



- Adding more layers to a suitably deep model leads to higher training error

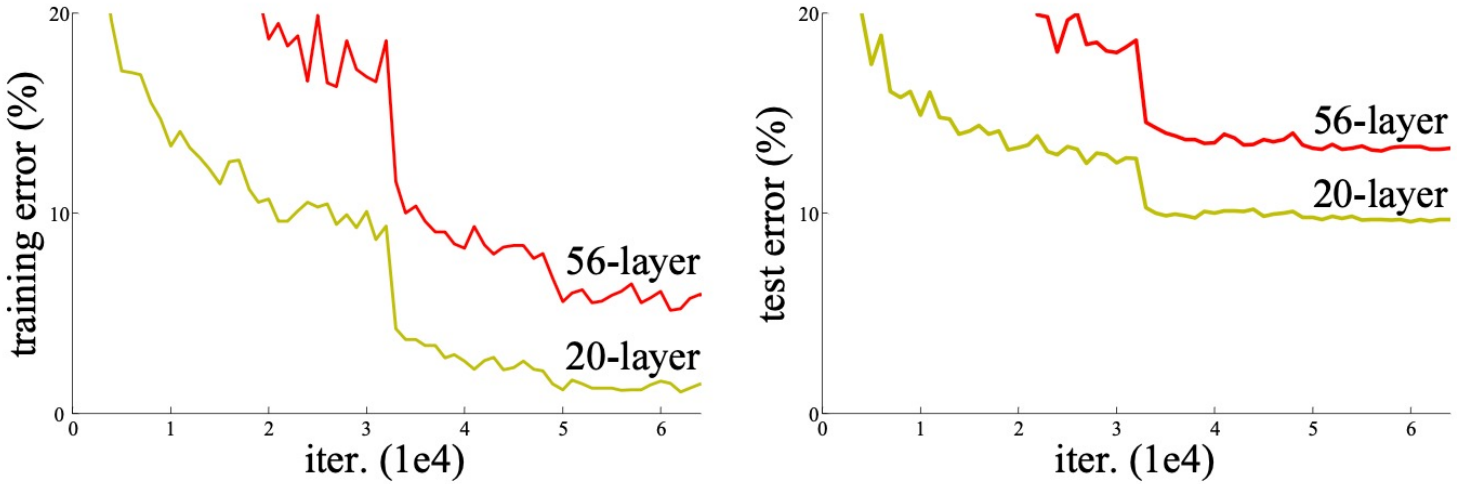



Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

# Introduction



- Not all systems are similarly easy to optimize
- Deep residual learning: Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping.
- Denoting the desired underlying mapping as $H(\mathbf{x})$, we let the stacked nonlinear layers fit another mapping of $F(\mathbf{x}) := H(\mathbf{x}) - \mathbf{x}$. The original mapping is recast into $F(\mathbf{x})+\mathbf{x}$.

# Introduction



- We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping.
- To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

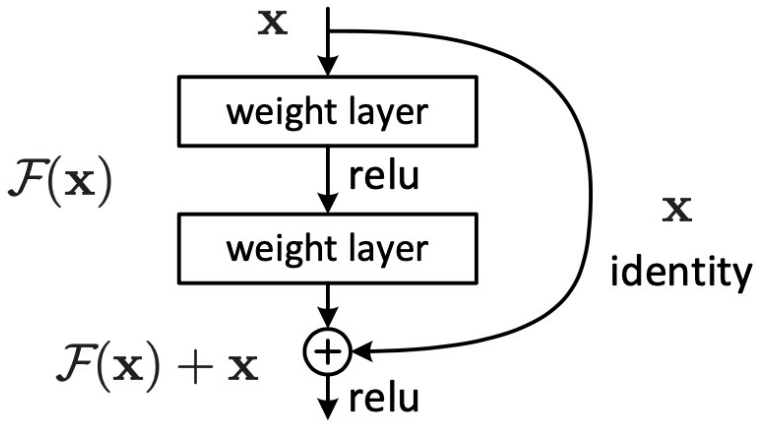



Figure 2. Residual learning: a building block.

# Introduction

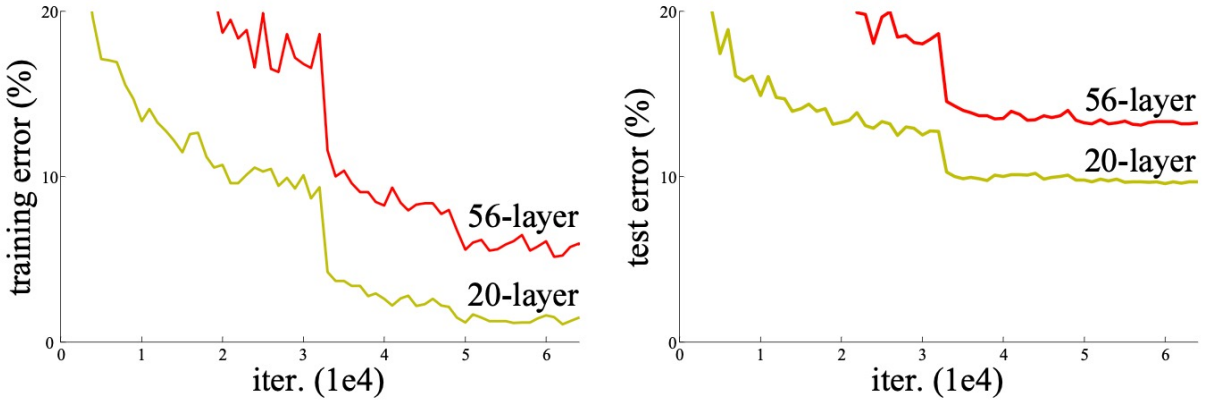


- Our extremely deep residual nets are easy to optimize.
- Our deep residual nets can easily enjoy accuracy gains from greatly increased depth, producing results substantially better than previous networks.
- Our 152-layer residual net is the deepest network ever presented on ImageNet, while still having lower complexity than VGG nets.

# Deep Residual Learning



- If the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart.
- The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers.



# Identity Mapping by Shortcuts



- We adopt residual learning to every few stacked layers.
- We consider a building block defined as:

$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

- Here **x** and **y** are the input and output vectors of the layers considered. The function $F(\mathbf{x}, \{W_i\})$ represents the residual mapping to be learned.

$$F = W_2\sigma(W_1\boldsymbol{x})$$

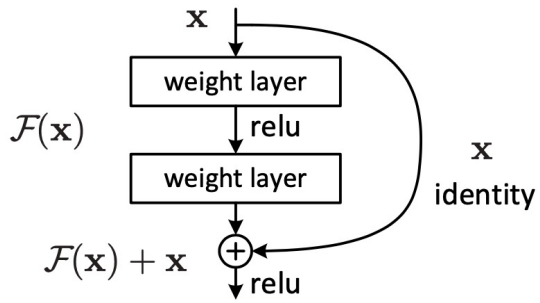



Figure 2. Residual learning: a building block.

# Identity Mapping by Shortcuts



- The operation $F + \mathbf{x}$ is performed by a shortcut connection and element-wise addition.
- We adopt the second nonlinearity after the addition (i.e., $\sigma(\mathbf{y})$, see Fig. 2).
- The shortcut connections introduce neither extra parameter nor computation complexity.

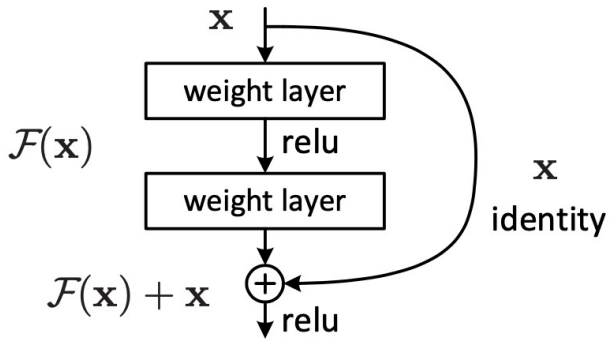



Figure 2. Residual learning: a building block.

# Identity Mapping by Shortcuts



- $\mathbf{x}$ and $F$ can be in different dimensions. We can perform a linear projection $W_s$ by the shortcut connections to match the dimensions:

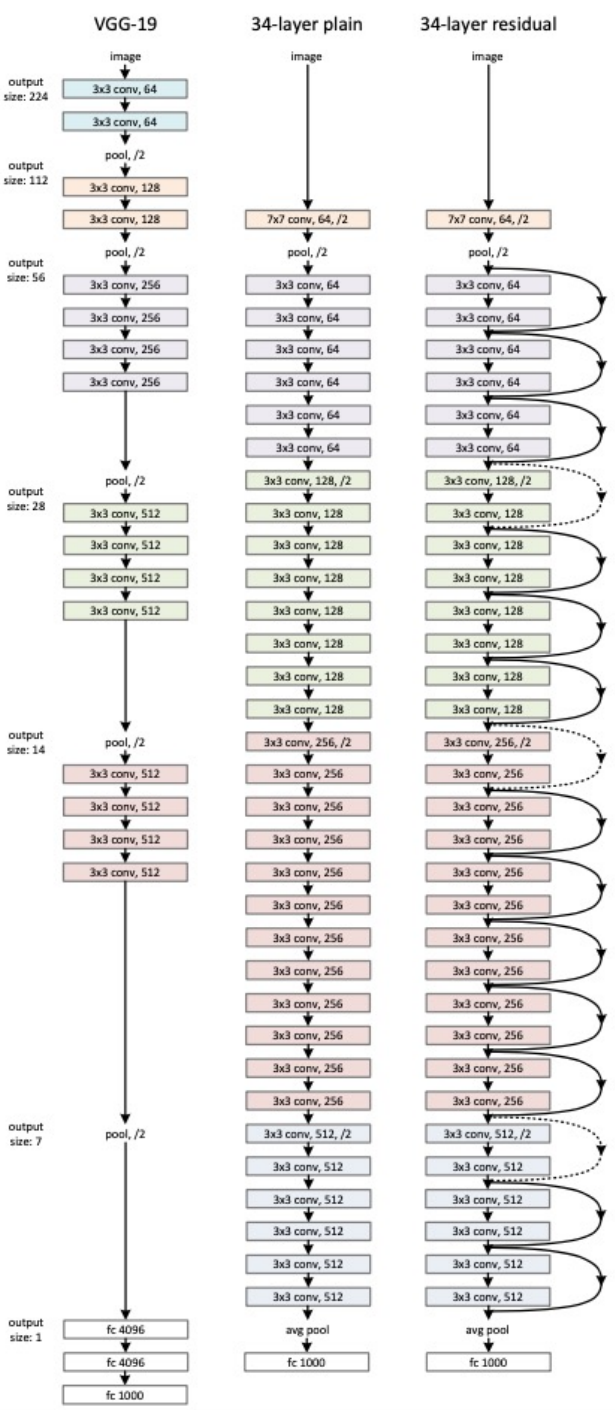
$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$$

- Although the above notations are about fully-connected layers for simplicity, they are applicable to convolutional layers.
- The function $F(\mathbf{x}, \{W_i\})$ can represent multiple convolutional layers. The element-wise addition is performed on two feature maps, channel by channel.

# Network Architectures



- VGG-19
- Plain network
  - 3.6 billion FLOPs (multiply-adds), only 18% of VGG-19 (19.6 billion FLOPs)
- Residual network
  - Add shortcuts to the plain network



# Implementation



- The image is resized with its shorter side randomly sampled in [256, 480] for scale augmentation.
- A 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted.
- Batch normalization (BN) right after each convolution and before activation
- SGD with a mini-batch size of 256
- The learning rate starts from 0.1 and is divided by 10 when the error plateaus
- The models are trained for up to $60 \times 10^4$ iterations
- Use a weight decay of 0.0001 and a momentum of 0.9.

# Implementation



- In testing, adopt the standard 10-crop testing.
- Average the scores at multiple scales (images are resized such that the shorter side is in {224, 256, 384, 480, 640}).

# Experiments



- Evaluate our method on the ImageNet 2012 classification dataset that consists of 1000 classes. The models are trained on the 1.28 million training images, and evaluated on the 50k validation images.
- We first evaluate 18-layer and 34-layer plain nets.

| | plain | ResNet |
|---|---|---|
| 18 layers | 27.94 | 27.88 |
| 34 layers | 28.54 | **25.03** |

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

# Experiments



- The 34-layer ResNet exhibits considerably lower training error and is generalizable to the validation data.
- ResNet eases the optimization by providing faster convergence at the early stage.

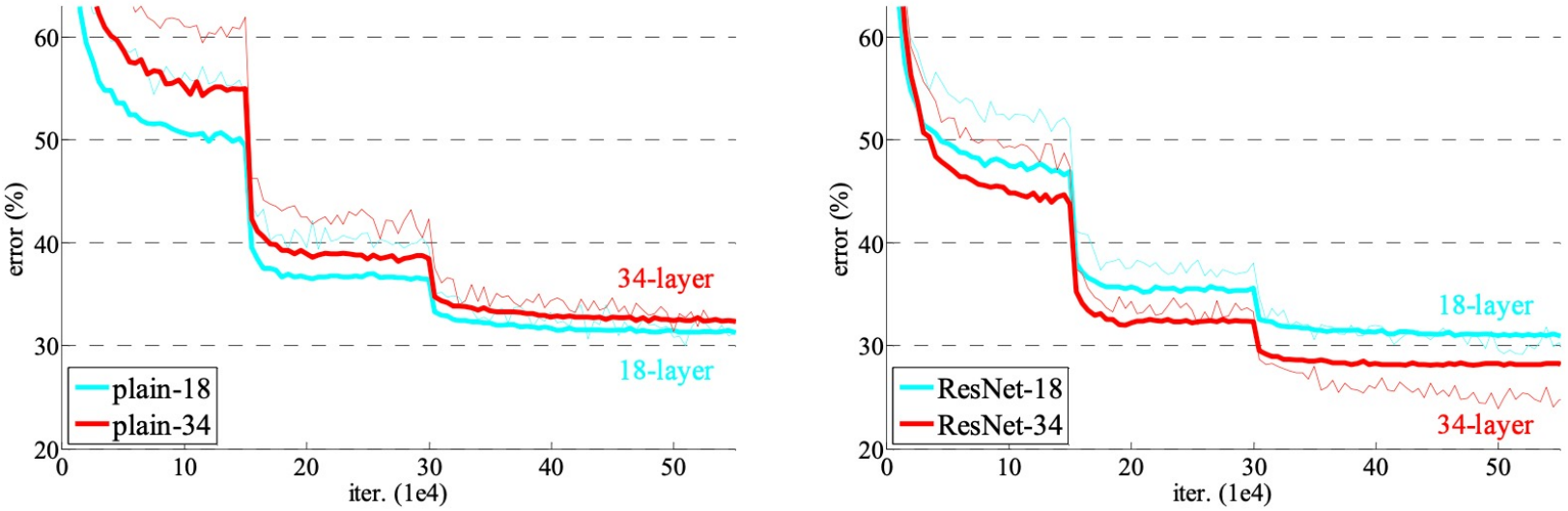



Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

# Experiments



- Identity vs. Projection Shortcuts
  - (A) zero-padding shortcuts are used for increasing dimensions, and all shortcuts are parameter-free
  - (B) projection shortcuts are used for increasing dimensions, and other shortcuts are identity
  - (C) all shortcuts are projections

| model | top-1 err. | top-5 err. |
|---|---|---|
| VGG-16 [40] | 28.07 | 9.33 |
| GoogLeNet [43] | - | 9.15 |
| PReLU-net [12] | 24.27 | 7.38 |
| plain-34 | 28.54 | 10.02 |
| ResNet-34 A | 25.03 | 7.76 |
| ResNet-34 B | 24.52 | 7.46 |
| ResNet-34 C | 24.19 | 7.40 |
| ResNet-50 | 22.85 | 6.71 |
| ResNet-101 | 21.75 | 6.05 |
| ResNet-152 | **21.43** | **5.71** |

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

# Experiments



- Deeper Bottleneck Architectures
  - We modify the building block as a bottleneck design
  - For each residual function $F$, we use a stack of 3 layers instead of 2. The three layers are $1\times1$, $3\times3$, and $1\times1$ convolutions, where the $1\times1$ layers are responsible for reducing and then increasing (restoring) dimensions, leaving the $3\times3$ layer a bottleneck with smaller input/output dimensions.

# Experiments



- Deeper Bottleneck Architectures

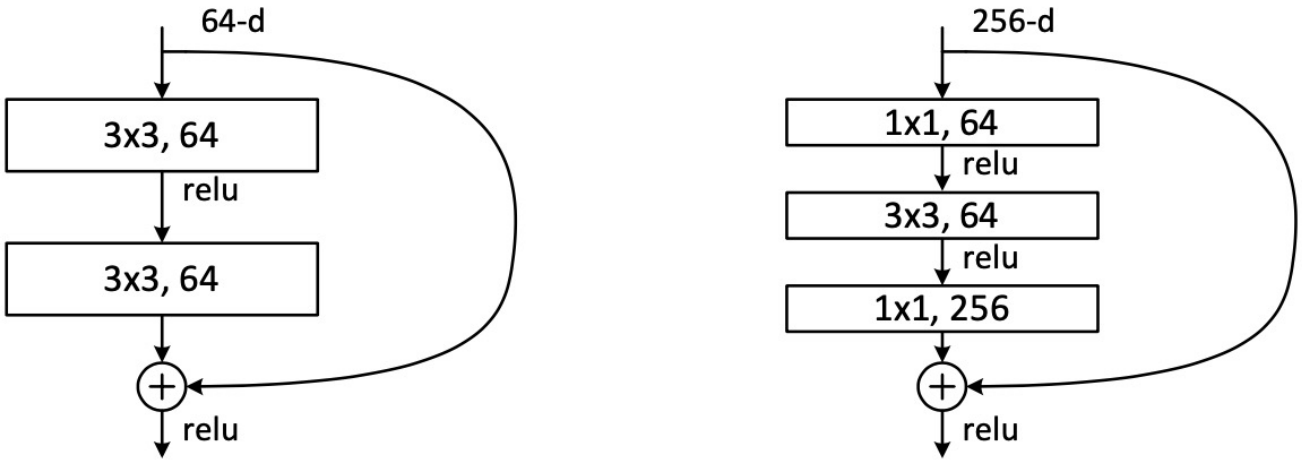



Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

# Experiments



- 50-layer ResNet: We replace each 2-layer block in the 34-layer net with this 3-layer bottleneck block, resulting in a 50-layer ResNet. We use option B for increasing dimensions. This model has 3.8 billion FLOPs.
- 101-layer and 152-layer ResNets: We construct 101-layer and 152-layer ResNets by using more 3-layer blocks.
- The 152-layer ResNet (11.3 billion FLOPs) still has lower complexity than VGG-16/19 nets (15.3/19.6 billion FLOPs).

# Experiments



| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

# Experiments



- 152-layer ResNet has a single-model top-5 validation error of 4.49%.

| method | top-1 err. | top-5 err. |
|---|---|---|
| VGG [40] (ILSVRC'14) | - | 8.43† |
| GoogLeNet [43] (ILSVRC'14) | - | 7.89 |
| VGG [40] (v5) | 24.4 | 7.1 |
| PReLU-net [12] | 21.59 | 5.71 |
| BN-inception [16] | 21.99 | 5.81 |
| ResNet-34 B | 21.84 | 5.71 |
| ResNet-34 C | 21.53 | 5.60 |
| ResNet-50 | 20.74 | 5.25 |
| ResNet-101 | 19.87 | 4.60 |
| ResNet-152 | **19.38** | **4.49** |

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

| method | top-5 err. (**test**) |
|---|---|
| VGG [40] (ILSVRC'14) | 7.32 |
| GoogLeNet [43] (ILSVRC'14) | 6.66 |
| VGG [40] (v5) | 6.8 |
| PReLU-net [12] | 4.94 |
| BN-inception [16] | 4.82 |
| **ResNet (ILSVRC'15)** | **3.57** |

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

# Experiments



- CIFAR-10 and Analysis
  - CIFAR-10 dataset consists of 50k training images and 10k testing images in 10 classes.

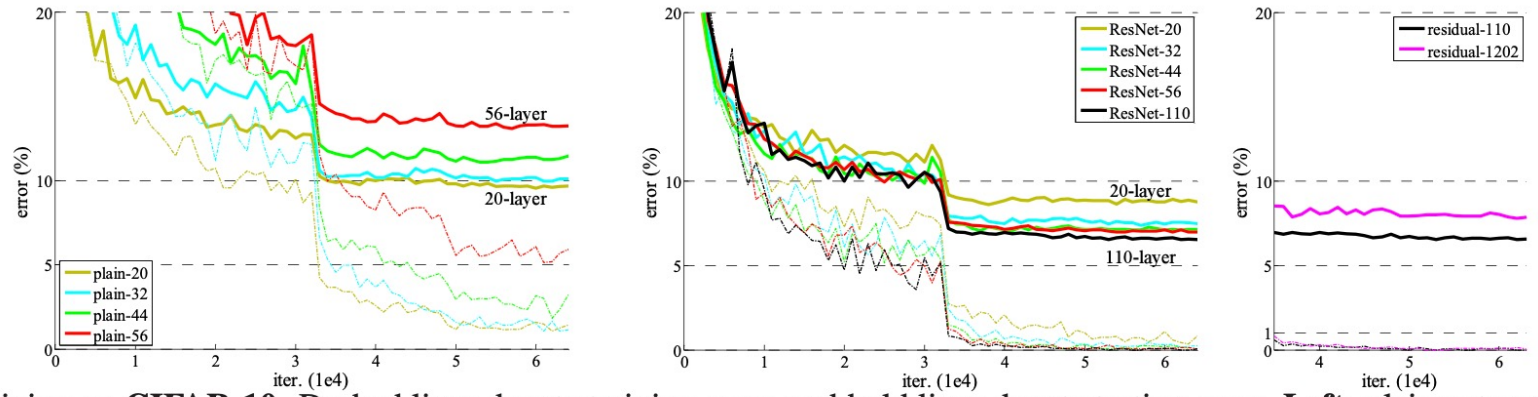



Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left**: plain networks. The error of plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1202 layers.

# Experiments



- Object Detection on PASCAL and MS COCO
  - We adopt Faster R-CNN as the detection method. Here we are interested in the improvements of replacing VGG-16 with ResNet-101.

| training data | 07+12 | 07++12 |
|---|---|---|
| test data | VOC 07 test | VOC 12 test |
| VGG-16 | 73.2 | 70.4 |
| ResNet-101 | **76.4** | **73.8** |

Table 7. Object detection mAP (%) on the PASCAL VOC 2007/2012 test sets using **baseline** Faster R-CNN. See also appendix for better results.

| metric | mAP@.5 | mAP@[.5, .95] |
|---|---|---|
| VGG-16 | 41.5 | 21.2 |
| ResNet-101 | **48.4** | **27.2** |

Table 8. Object detection mAP (%) on the COCO validation set using **baseline** Faster R-CNN. See also appendix for better results.