Trade-offs in architecture

Between filter size and number of layers
- Keep the layer widths fixed.
- Deeper networks with smaller filter sizes perform better (if you keep the overall computational complexity fixed)

Between layer width and number of layers
- Keep the size of the filters fixed.
- Increasing depth improves performance

Between filter size and layer width
- Keep the number of layers fixed.
- No significant difference
Main sources of memory load:

- **Activation maps:**
  - Training: They need to be kept during training so that backpropagation can be performed
  - Testing: No need to keep the activations of earlier layers

- **Parameters:**
  - The weights, their gradients and also another copy if momentum is used

- **Data:**
  - The originals + their augmentations

- If all these don’t fit into memory,
  - Load your data batch by batch from disk
  - Decrease the size of your batches
Finetuning

1. If the new dataset is small and similar to the original dataset used to train the CNN:
   – Finetuning the whole network may lead to overfitting
   – Just train the newly added layer

2. If the new dataset is big and similar to the original dataset:
   – The more, the merrier: go ahead and train the whole network

3. If the new dataset is small and different from the original dataset:
   – Not a good idea to train the whole network
   – However, add your new layer not to the top of the network, since those parts are very dataset (problem) specific
   – Add your layer to earlier parts of the network

4. If the new dataset is big and different from the original dataset:
   – We can “finetune” the whole network
   – This amounts to a new training problem by initializing the weights with those of another network
Visualize activations during training

- Activations are dense at the beginning.
  - They should get sparser during training.
- If some activation maps are all zero for many inputs, dying neuron problem => high learning rate in the case of ReLUs.

http://cs231n.github.io/convolutional-networks/
Visualize the weights

• We can directly look at the filters of all layers
• First layer is easier to interpret
• Filters shouldn’t look noisy
Visualize the inputs that maximally activate a neuron

Keep track of which images activate a neuron most

Maximally activating images for some POOL5 (5th pool layer) neurons of an AlexNet. The activation values and the receptive field of the particular neuron are shown in white. (In particular, note that the POOL5 neurons are a function of a relatively large portion of the input image!) It can be seen that some neurons are responsive to upper bodies, text, or specular highlights.
Embed the codes in a lower-dimensional space

- Place images into a 2D space such that images which produce similar CNN codes are placed close.
- You can use, e.g., t-Distributed Stochastic Neighbor Embedding (t-SNE)

![t-SNE embedding of a set of images based on their CNN codes. Images that are nearby each other are also close in the CNN representation space, which implies that the CNN "sees" them as being very similar. Notice that the similarities are more often class-based and semantic rather than pixel and color-based. For more details on how this visualization was produced the associated code, and more related visualizations at different scales refer to t-SNE visualization of CNN codes.]

Figure 1: Illustration of t-SNE on MNIST dataset

Figure: Laurens van der Maaten and Geoffrey Hinton
Occlude parts of the image

• Slide an “occlusion window” over the image
• For each occluded image, determine the class prediction confidence/probability.

Three input images (top). Notice that the occluder region is shown in grey. As we slide the occluder over the image we record the probability of the correct class and then visualize it as a heatmap (shown below each image). For instance, in the left-most image we see that the probability of Pomeranian plummets when the occluder covers the face of the dog, giving us some level of confidence that the dog’s face is primarily responsible for the high classification score. Conversely, zeroing out other parts of the image is seen to have relatively negligible impact.
Data gradients

Generate an image that maximizes the class score.

More formally, let $S_c(I)$ be the score of the class $c$, computed by the classification layer of the ConvNet for an image $I$. We would like to find an $L_2$-regularised image, such that the score $S_c$ is high:

$$\arg \max_I S_c(I) - \lambda \|I\|_2^2,$$

where $\lambda$ is the regularisation parameter. A locally-optimal $I$ can be found by the back-propagation

• Use: Gradient ascent!
Data gradients

The gradient with respect to the input is high for pixels which are on the object.

We start with a motivational example. Consider the linear score model for the class \( c \):

\[
S_c(I) = w_c^T I + b_c,
\]

(2)

where the image \( I \) is represented in the vectorised (one-dimensional) form, and \( w_c \) and \( b_c \) are respectively the weight vector and the bias of the model. In this case, it is easy to see that the magnitude of elements of \( w \) defines the importance of the corresponding pixels of \( I \) for the class \( c \).

In the case of deep ConvNets, the class score \( S_c(I) \) is a highly non-linear function of \( I \), so the reasoning of the previous paragraph cannot be immediately applied. However, given an image \( I_0 \), we can approximate \( S_c(I) \) with a linear function in the neighbourhood of \( I_0 \) by computing the first-order Taylor expansion:

\[
S_c(I) \approx w^T I + b,
\]

(3)

where \( w \) is the derivative of \( S_c \) with respect to the image \( I \) at the point (image) \( I_0 \):

\[
w = \left. \frac{\partial S_c}{\partial I} \right|_{I_0}.
\]

(4)

Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps

Karen Simonyan
Andrea Vedaldi
Andrew Zisserman

Visual Geometry Group, University of Oxford

(karen,vedaldi,az)@robots.ox.ac.uk

2014
Class Activation Maps

Weighted combination of the feature maps before GAP:

\[ M(x, y) = \sum_k w_k^c f_k(x, y) \]

Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

Class Activation Maps

GradCAM:

\[ \alpha^c_k = \sum_{x,y} \frac{\partial S_c}{\partial f_k} \]

\[ M^c(x, y) = ReLU \left( \sum_k \alpha^c_k f_k(x, y) \right) \]


Figure: https://pypi.org/project/grad-cam/

<table>
<thead>
<tr>
<th>Network</th>
<th>Image</th>
<th>GradCAM</th>
<th>GradCAM++</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td><img src="image1.png" alt="VGG16 Image" /></td>
<td><img src="image2.png" alt="VGG16 GradCAM" /></td>
<td><img src="image3.png" alt="VGG16 GradCAM++" /></td>
</tr>
<tr>
<td>Resnet50</td>
<td><img src="image4.png" alt="Resnet50 Image" /></td>
<td><img src="image5.png" alt="Resnet50 GradCAM" /></td>
<td><img src="image6.png" alt="Resnet50 GradCAM++" /></td>
</tr>
</tbody>
</table>
Feature inversion

• Learns to reconstruct an image from its representation

This section introduces our method to compute an approximate inverse of an image representation. This is formulated as the problem of finding an image whose representation best matches the one given [34]. Formally, given a representation function $\Phi : \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}^d$ and a representation $\Phi_0 = \Phi(x_0)$ to be inverted, reconstruction finds the image $x \in \mathbb{R}^{H \times W \times C}$ that minimizes the objective:

$$x^* = \arg\min_{x \in \mathbb{R}^{H \times W \times C}} \ell(\Phi(x), \Phi_0) + \lambda \mathcal{R}(x)$$  \hspace{1cm} (1)

where the loss $\ell$ compares the image representation $\Phi(x)$ to the target one $\Phi_0$ and $\mathcal{R} : \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}$ is a regulariser capturing a natural image prior.

• Regularization term here is the key factor, e.g. a combination of the two terms:

$$\mathcal{R}_\alpha(x) = \|x\|_{\alpha}$$

$$\mathcal{R}_{\nu,\beta}(x) = \sum_{i,j} \left( (x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

Spring 2021

Previously on CENG501!
Feature inversion with perceptual losses

Figure from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016.
Fooling ConvNets

- Given an image $I$ labeled as $l_1$, find minimum “$r$” (noise) such that $I + r$ is classified as a different label, $l_2$.
- I.e., minimize:
  
  $\arg \min_r \text{loss}(I + r, l_2) + c |r|$

---

Previously on CENG501!
Today

- Convolutional Neural Networks
  - Widely-used CNN architectures
  - CNN Applications

- Recurrent Neural Networks

---

These slides available at: [https://kovan.ceng.metu.edu.tr/~sinan/DL/week_12.pdf](https://kovan.ceng.metu.edu.tr/~sinan/DL/week_12.pdf)
Administrative Issues

- Programming assignment 1
- Take-Home Exam 1

- Office Hour:
  - Every Tuesday, 21:00

- Project paper selection
  - [Link](https://docs.google.com/spreadsheets/d/1tzPHq_Vgu6gCwNyXJHGvqeA6p gU67H0nKYjqkisWfKc/edit?usp=sharing)
  - Deadline: 19th of April
POPULAR
CNN MODELS
LeNet (1998)

- For reading zip codes and digits

Euclidean RBF:

\[ y_i = \sum_j (x_j - w_{ij})^2. \]
AlexNet (2012)

- Popularized CNN in computer vision & pattern recognition
- ImageNet ILSVRC challenge 2012 winner
- Similar to LeNet
  - Deeper & bigger
  - Many CONV layers on top of each other (rather than adding immediately a pooling layer after a CONV layer)
  - Uses GPU
- 650K neurons. 60M parameters. Trained on 2 GPUs for a week.
AlexNet (2012) Details

- Since the network is too big to fit in on GPU, it is divided into two.
- Note the cross connections between the “pathways”.
- Uses ReLU as non-linearity after every convolutional and fully-connected layer.
- Normalization layer is placed after the first & the second convolutional layers.
- Max-pooling layer is placed only after the normalization layers & the fifth convolutional layer.
- Last layer is a soft-max.
AlexNet (2012) Training

- Data augmentation & dropout are used during training to avoid overfitting.
- Stochastic Gradient Descent with a batchsize of 128 examples is used.
- Momentum with coefficient 0.9 is employed.
- Weight decay (L2 regularization cost) with factor 0.0005 is also used in the loss function.
- Weights are initialized from a zero-mean Gaussian distribution with 0.01 std.
- Learning rate started with 0.01 and manually divided by 10 when the validation error rate stopped improving.
- Trained on 1.2 million images, which took 5-6 days on two GPUs.
AlexNet (2012): **The learned filters**

- Do you notice anything strange with the filters?

![Figure 3: 96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 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.](image)

**Figure 3** shows the convolutional kernels learned by the network’s two data-connected layers. The network has learned a variety of frequency- and orientation-selective kernels, as well as various colored blobs. Notice the specialization exhibited by the two GPUs, a result of the restricted connectivity described in Section 3.5. The kernels on GPU 1 are largely color-agnostic, while the kernels on GPU 2 are largely color-specific. This kind of specialization occurs during every run and is independent of any particular random weight initialization (modulo a renumbering of the GPUs).
GoogleNet (2014)

• ImageNet 2014 winner
• Contributions:
  – Inception module
    • Dramatically reduced parameters (from 60M in AlexNet to 4M)
  – Avg Pooling at the top, instead of fully-connected layer ➔ Reduced number of parameters
• Motivation:
  – Going bigger (in depth or width) means too many parameters.
  – Go bigger by maintaining sparse connections.
Inception module: “network in network” (inspired from Lin et al., 2013)

- Concatenation is performed along the “columns” (depth).
  - The output of inception layers must have the same size.
- The naïve version has a tendency to blow up in number of channels.
  - Why? Max-pooling does not change the number of channels. When concatenated with other filter responses, number of channels increase with every layer.
  - Solution: Do 1x1 convolution to decrease the number of channels.
    - Also called “bottleneck”.
- In order to decrease the computational complexity of 3x3 and 5x5 pooling, they are also preceded by 1x1 convolution (i.e., the number of channels are reduced).
One of the main beneficial aspects of this architecture is that it allows for increasing the number of units at each stage significantly without an uncontrolled blow-up in computational complexity. The ubiquitous use of dimension reduction allows for shielding the large number of input filters of the last stage to the next layer, first reducing their dimension before convolving over them with a large patch size. Another practically useful aspect of this design is that it aligns with the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously.
• Since the intermediate layers learn to discriminate features specific to a class, we can directly link them to the loss term.
  – Encourages these layers to become more discriminative
  – Increases propagation of gradient signal to earlier stages
GoogleNet: More Details

• ReLU after all layers
• Max pooling in inception modules as well as a whole layer occasionally
• Avg pooling instead of fully-connected layers
  – Only a minor change in the accuracy (0.6%)
  – However, less number of parameters
• Other usual tricks (e.g., dropout, augmentation etc.) are used.
• Trained on CPUs using a distributed machine learning system.
• SGD with momentum (0.9).
• Fixed learning rate scheme with 4% decrease every 8 epochs
• They trained many different models with different initializations and parameters. They combined these models using different methods and tricks. There is no single training method that yields the results they achieved.
VGGNet (2014)

• ImageNet runner up in 2014
• Contribution:
  – Use small RFs & increase depth as much as possible
  – 16 CONV/FC layers.
  – 3x3 CONVs and 2x2 pooling from beginning to the end
• Although performs slightly worse than GoogleNet in image classification, VGGNet may perform better at other tasks (such as transfer learning problems).
• Downside: Needs a lot of memory & parameters (140M)
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.

<table>
<thead>
<tr>
<th></th>
<th>A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11 weight layers</td>
<td>11 weight layers</td>
<td>13 weight layers</td>
<td>16 weight layers</td>
<td>16 weight layers</td>
</tr>
<tr>
<td>input (224 × 224 RGB image)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
</tr>
<tr>
<td>LRN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
</tr>
<tr>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-128</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC-4096</td>
<td>FC-4096</td>
<td>FC-1000</td>
<td>soft-max</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Number of parameters (in millions).

<table>
<thead>
<tr>
<th>Network</th>
<th>A-A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of parameters</td>
<td>133</td>
<td>133</td>
<td>134</td>
<td>138</td>
<td>144</td>
</tr>
</tbody>
</table>

- Increasing the depth naively may not give you better performance after a number of depths
  - Why?
    - This is shown to be not due to overfitting (since training error also gets worse) or vanishing gradients (suitable non-linearities used)
    - Accuracy is somehow saturated. Not clear why. Though reported in several studies.
- Solution: Make shortcut connections

- Residual (shortcut) connections

\[ F(x) + x \]

Figure 2. Residual learning: a building block.

Figure 5. A deeper residual function \( F \) 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.

- Residual (shortcut) connections

<table>
<thead>
<tr>
<th>method</th>
<th>top-1 err.</th>
<th>top-5 err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG [41] (ILSVRC’14)</td>
<td>-</td>
<td>8.43*</td>
</tr>
<tr>
<td>GoogLeNet [44] (ILSVRC’14)</td>
<td>-</td>
<td>7.89</td>
</tr>
<tr>
<td>VGG [41] (v5)</td>
<td>24.4</td>
<td>7.1</td>
</tr>
<tr>
<td>BN-inception [16]</td>
<td>21.99</td>
<td>5.81</td>
</tr>
<tr>
<td>ResNet-34 B</td>
<td>21.84</td>
<td>5.71</td>
</tr>
<tr>
<td>ResNet-34 C</td>
<td>21.53</td>
<td>5.60</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>20.74</td>
<td>5.25</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>19.87</td>
<td>4.60</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>19.38</td>
<td>4.49</td>
</tr>
</tbody>
</table>

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

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.
Effect of residual connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The vertical axis is logarithmic to show dynamic range. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Visualizing the loss landscape of neural nets

Hao Li¹, Zheng Xu¹, Gavin Taylor², Christoph Studer³, Tom Goldstein¹
¹University of Maryland, College Park, ²United States Naval Academy, ³Cornell University
{haoli, xuzh, tomg}@cs.umd.edu, taylor@usna.edu, studer@cornell.edu

2018
Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, Jeffrey Pennington

Abstract

In recent years, state-of-the-art methods in computer vision have utilized increasingly deep convolutional neural network architectures (CNNs), with some of the most successful models employing hundreds or even thousands of layers. A variety of pathologies such as vanishing/exploding gradients make training such deep networks challenging. While residual connections and batch normalization do enable training at these depths, it has remained unclear whether such specialized architecture designs are truly necessary to train deep CNNs. In this work, we demonstrate that it is possible to train vanilla CNNs with ten thousand layers or more simply by using an appropriate initialization scheme. We derive this initialization scheme theoretically by developing a mean field theory for signal propagation and by characterizing the conditions for dynamical isometry, the equilibration of singular values of the input-output Jacobian matrix. These conditions require that the convolution operator be an orthogonal transformation in the sense that it is norm-preserving. We present an algorithm for generating such random initial orthogonal convolution kernels and demonstrate empirically that they enable efficient training of extremely deep architectures.

"Our initial results suggest that past a certain depth, on the order of tens or hundreds of layers, the test performance for vanilla convolutional architecture saturates. These observations suggest that architectural features such as residual connections and batch normalization are likely to play an important role in defining a good model class, rather than simply enabling efficient training."
DiracNets: Training Very Deep Neural Networks Without Skip-Connections

Sergey Zagoruyko
sergey.zagoruyko@enpc.fr
Université Paris-Est, École des Ponts
ParisTech
Paris, France

Nikos Komodakis
nikos.komodakis@enpc.fr
Paristech
Paris, France

Abstract

Deep neural networks with skip-connections, such as ResNet, show excellent performance in various image classification benchmarks. It is thought observed that the initial motivation behind them - training deeper networks - does not actually hold true, and the benefits come from increased capacity, rather than from depth. Motivated by this, and inspired from ResNet, we propose a simple Dirac weight parameterization, which allows us to train very deep plain networks without skip-connections, and achieve nearly the same performance. This parameterization has a minor computational cost at training time and no cost at all at inference. We’re able to achieve 95.5% accuracy on CIFAR-10 with 34-layer deep plain network, surpassing 1001-layer deep ResNet, and approaching Wide ResNet. Our parameterization also mostly eliminates the need of careful initialization in residual and non-residual networks. The code and models for our experiments are available at https://github.com/szagoruyko/diracnets
RepVGG: Making VGG-style ConvNets Great Again

Xiaohan Ding 1*  Xiangyu Zhang 2  Ningning Ma 3
Jungong Han 4  Guiguang Ding 1†  Jian Sun 2

1 Beijing National Research Center for Information Science and Technology (BNRist);
School of Software, Tsinghua University, Beijing, China
2 MEGVII Technology
3 Hong Kong University of Science and Technology
4 Computer Science Department, Aberystwyth University, SY23 3FL, UK

dxh17@mails.tsinghua.edu.cn  zhangxiangyu@megvii.com  nmaac@cse.ust.hk
jungonghan77@gmail.com  dinggg@tsinghua.edu.cn  sunjian@megvii.com

Abstract
ResNext

Figure 1. **Left:** A block of ResNet [14]. **Right:** A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).
Densely Connected Convolutional Networks

Gao Huang*  
Cornell University  
g34@cornell.edu

Zhuang Liu*  
Tsinghua University  
luzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten  
Facebook AI Research  
lvdmaaten@fb.com

Kilian Q. Weinberger  
Cornell University  
kq4@cornell.edu

2016;2018

**Figure 3:** Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (left) and FLOPs during test-time (right).
Highway networks

• This is a regular MLP with gated units.

\[ y = H(x, W_H). \]  
(1)

\( H \) is usually an affine transform followed by a non-linear activation function, but in general it may take other forms.

For a highway network, we additionally define two non-linear transforms \( T(x, W_T) \) and \( C(x, W_C) \) such that

\[ y = H(x, W_H) \cdot T(x, W_T) + x \cdot C(x, W_C). \]  
(2)

We refer to \( T \) as the transform gate and \( C \) as the carry gate, since they express how much of the output is produced by transforming the input and carrying it, respectively. For simplicity, in this paper we set \( C = 1 - T \), giving

\[ y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T)). \]  
(3)

The dimensionality of \( x, y, H(x, W_H) \) and \( T(x, W_T) \) must be the same for Equation (3) to be valid. Note that this re-parametrization of the layer transformation is much more flexible than Equation (1). In particular, observe that

\[ y = \begin{cases} x, & \text{if } T(x, W_T) = 0, \\ H(x, W_H), & \text{if } T(x, W_T) = 1. \end{cases} \]  
(4)
Highway Networks

https://www.researchgate.net/publication/311842587_Highway_and_Residual_Networks_learn_Unrolled_Iterative_Estimation
Comparison:
Comparison:

Going deep may not be the only answer

Shallow Networks for High-Accuracy Road Object-Detection

Khalid Ashraf, Bichen Wu, Forrest N. Iandola, Matthew W. Moskewicz, Kurt Keutzer
Electrical Engineering and Computer Sciences Department, UC Berkeley
{ashrafkhalid, bichen}@berkeley.edu, {forresti, moskewicz, keutzer}@eecs.berkeley.edu

Abstract

The ability to automatically detect other vehicles on the road is vital to the safety of partially-autonomous and fully-autonomous vehicles. Most of the high-accuracy techniques for this task are based on R-CNN or one of its faster variants. In the research community, much emphasis has been applied to using 3D vision or complex R-CNN variants to achieve higher accuracy. However, are there more straightforward modifications that could deliver higher accuracy? Yes. We show that increasing input image resolution (i.e. upsampling) offers up to 12 percentage-points higher accuracy compared to an off-the-shelf baseline. We also find situations where earlier/shallower layers of CNN provide higher accuracy than later/deeper layers. We further show that shallow models and upsampled images yield competitive accuracy. Our findings contrast with the current trend towards deeper and larger models to achieve high accuracy in domain specific detection tasks.
Recent work


- Major winning Convolutional Neural Networks (CNNs), such as VGGNet, ResNet, DenseNet, etc, include tens to hundreds of millions of parameters, which impose considerable computation and memory overheads. This limits their practical usage in training and optimizing for real-world applications. On the contrary, light-weight architectures, such as SqueezeNet, are being proposed to address this issue. However, they mainly suffer from low accuracy, as they have compromised between the processing power and efficiency. These inefficiencies mostly stem from following an ad-hoc designing procedure. In this work, we discuss and propose several crucial design principles for an efficient architecture design and elaborate intuitions concerning different aspects of the design procedure. Furthermore, we introduce a new layer called SAF-pooling to improve the generalization power of the network while keeping it simple by choosing best features. Based on such principles, we propose a simple architecture called SimpNet. We empirically show that SimpNet provides a good trade-off between the computation/memory efficiency and the accuracy solely based on these primitive but crucial principles. SimpNet outperforms the deeper and more complex architectures such as VGGNet, ResNet, WideResidualNet etc, on several well-known benchmarks, while having 2 to 25 times fewer number of parameters and operations. We obtain state-of-the-art results (in terms of a balance between the accuracy and the number of involved parameters) on standard datasets, such as CIFAR10, CIFAR100, MNIST and SVHN. The implementations are available at this https URL.

https://arxiv.org/abs/1802.06205
Binary networks

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks

Mohammad Rastegari†, Vicente Ordonez‡, Joseph Redmon*, Ali Farhadi†

Allen Institute for AI†, University of Washington*
{mohammadr, vicenteor}@allenai.org
{pjrreddie, ali}@cs.washington.edu

Abstract. We propose two efficient approximations to standard convolutional neural networks: Binary-Weight-Networks and XNOR-Networks. In Binary-Weight-Networks, the filters are approximated with binary values resulting in \(32 \times\) memory saving. In XNOR-Networks, both the filters and the input to convolutional layers are binary. XNOR-Networks approximate convolutions using primarily binary operations. This results in \(58 \times\) faster convolutional operations and \(32 \times\) memory savings. XNOR-Nets offer the possibility of running state-of-the-art networks on CPUs (rather than GPUs) in real-time. Our binary networks are simple, accurate, efficient, and work on challenging visual tasks. We evaluate our approach on the ImageNet classification task. The classification accuracy with a Binary-Weight-Network version of AlexNet is only 2.9% less than the full-precision AlexNet (in top-1 measure). We compare our method with recent network binarization methods, BinaryConnect and BinaryNets, and outperform these methods by large margins on ImageNet, more than 16% in top-1 accuracy.
Fig. 1: We propose two efficient variations of convolutional neural networks. **Binary-Weight-Networks**, when the weight filters contain binary values. **XNOR-Networks**, when both weight and input have binary values. These networks are very efficient in terms of memory and computation, while being very accurate in natural image classification. This offers the possibility of using accurate vision techniques in portable devices with limited resources.
Exploring Randomly Wired Neural Networks for Image Recognition

Saining Xie  Alexander Kirillov  Ross Girshick  Kaiming He
Facebook AI Research (FAIR)

Abstract

Neural networks for image recognition have evolved through extensive manual design from simple chain-like models to structures with multiple wiring paths. The success of ResNets [11] and DenseNets [16] is due in large part to their innovative wiring plans. Now, neural architecture search (NAS) studies are exploring the joint optimization of wiring and operation types, however, the space of possible wirings is constrained and still driven by manual design despite being searched. In this paper, we explore a more diverse set of connectivity patterns through the lens of randomly wired neural networks. To do this, we first define the concept of a stochastic network generator that encapsulates the entire network generation process. Encapsulation provides a unified view of NAS and randomly wired networks. Then, we use three classical random graph models to generate randomly wired graphs for networks. The results are surprising: several variants of these random generators yield network instances that have competitive accuracy on the ImageNet benchmark. These results suggest that new efforts focusing on designing better network generators may lead to new breakthroughs by exploring less constrained search spaces with more room for novel design.

Figure 1. Randomly wired neural networks generated by the classical Watts-Strogatz (WS) [50] model; these three instances of random networks achieve (left-to-right) 79.1%, 79.1%, 79.0% classification accuracy on ImageNet under a similar computational budget to ResNet-50, which has 77.1% accuracy.
CNN APPLICATIONS
Object Detection: Regions with CNN (R-CNN)

- A very straightforward application
- Start from a pre-trained model (trained on imagenet)
- Finetune using the new data available

- There are faster versions (called Fast-RCNN) by sharing computations performed in convolutions on different regions

Fig. 1. Object detection system overview. Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional network (CNN), and then (4) classifies each region using class-specific linear SVMs. We trained an R-CNN that achieves a mean average precision (mAP) of 62.9% on PASCAL VOC 2010. For comparison, [21] reports 35.1% mAP using the same region proposals, but with a spatial pyramid and bag-of-visual-words approach. The popular deformable part models perform at 33.4%. On the 200-class ILSVRC2013 detection dataset, we trained an R-CNN with a mAP of 31.4%, a large improvement over OverFeat [19], which had the previous best result at 24.3% mAP.
Faster R-CNN

- Replaced region-proposal method with a network
- Followed by the “Fast R-CNN” classifier
- The two networks share some convolutional layers

Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the ‘attention’ of this unified network.
RoI Pooling

• Goal: obtain fixed-size features from regions of different sizes

Figure: https://deepsense.ai/region-of-interest-pooling-explained/
Object Detection: Mask R-CNN

Mask R-CNN 2017

Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick
Facebook AI Research (FAIR)

Figure 1. The Mask R-CNN framework for instance segmentation.

Figure 2. Mask R-CNN results on the COCO test set. These results are based on ResNet-101 [19], achieving a mask AP of 35.7 and running at 5 fps. Masks are shown in color, and bounding box, category, and confidences are also shown.
You Look Only Once (YOLO)

Single-stage method:

• Perform classification & localization for a fixed set of potential regions.

Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to $448 \times 448$, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model’s confidence.

Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts $B$ bounding boxes, confidence for those boxes, and $C$ class probabilities. These predictions are encoded as an $S \times S \times (B + 5 + C)$ tensor.
Single-Shot Detector (SSD)

- Similar to YOLO, SSD is a single-stage detector
- Allows more bounding boxes per location
- 8372 BBs per class per image
Feature Pyramid Networks for Object Detection

- Used for feature extraction
- Can be used for region proposal, object detection or segmentation.

Figure 4. FPN for object segment proposals. The feature pyramid is constructed with identical structure as for object detection. We
Figure 3. The one-stage RetinaNet network architecture uses a Feature Pyramid Network (FPN) [20] backbone on top of a feedforward ResNet architecture [16] (a) to generate a rich, multi-scale convolutional feature pyramid (b). To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes (c) and one for regressing from anchor boxes to ground-truth object boxes (d). The network design is intentionally simple, which enables this work to focus on a novel focal loss function that eliminates the accuracy gap between our one-stage detector and state-of-the-art two-stage detectors like Faster R-CNN with FPN [20] while running at faster speeds.
Focal Loss

from the cross entropy (CE) loss for binary classification:

\[
CE(p, y) = \begin{cases} 
- \log(p) & \text{if } y = 1 \\
- \log(1 - p) & \text{otherwise.}
\end{cases} 
\] (1)

In the above, \( y \in \{\pm 1\} \) specifies the ground-truth class and \( p \in [0, 1] \) is the model's estimated probability for the class with label \( y = 1 \). For notational convenience, we define \( p_t \):

\[
p_t = \begin{cases} 
p & \text{if } y = 1 \\
1 - p & \text{otherwise},
\end{cases}
\] (2)

and rewrite \( CE(p, y) = CE(p_t) = - \log(p_t) \).

More formally, we propose to add a modulating factor \((1 - p_t)^\gamma\) to the cross entropy loss, with tunable focusing parameter \( \gamma \geq 0 \). We define the focal loss as:

\[
FL(p_t) = -(1 - p_t)^\gamma \log(p_t).
\] (4)

![Graph showing the comparison between CE and FL with different values of \( \gamma \). The graph illustrates how the focal loss FL penalizes well-classified examples less than CE.]

Figure 1. We propose a novel loss we term the Focal Loss that adds a factor \((1 - p_t)^\gamma\) to the standard cross entropy criterion. Setting \( \gamma > 0 \) reduces the relative loss for well-classified examples \((p_t > .5)\), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.
Takes 19 layer VGG as the base (no FC layers)

Max pooling is replaced by avg pooling since it produced more appealing results
Figure 1: Convolutional Neural Network (CNN). A given input image is represented as a set of filtered images at each processing stage in the CNN. While the number of different filters increases along the processing hierarchy, the size of the filtered images is reduced by some downsampling mechanism (e.g., max-pooling) leading to a decrease in the total number of units per layer of the network. **Content Reconstructions.** We can visualise the information at different processing stages in the CNN by reconstructing the input image from only knowing the network’s responses in a particular layer. We reconstruct the input image from convolution layers ‘conv1.1’ (a), ‘conv2.1’ (b), ‘conv3.1’ (c), ‘conv4.1’ (d) and ‘conv5.1’ (e) of the original VGG-Net. We find that reconstruction from lower layers is almost perfect (a,b,c). In higher layers of the network, detailed pixel information is lost while the high-level content of the image is preserved (d,e). **Style Reconstructions.** On top of the original CNN representations we built a new feature space that captures the style of an input image. The style representation computes correlations between the different features in different layers of the CNN. We reconstruct the style of the input image from style representations built on different subsets of CNN layers (‘conv1.1’ (a), ‘conv2.1’ (b) and ‘conv3.1’ (e), ‘conv1.1’, ‘conv2.1’, ‘conv3.1’ and ‘conv4.1’ (d), ‘conv1.1’, ‘conv2.1’, ‘conv3.1’, ‘conv4.1’ and ‘conv5.1’ (e)). This creates images that match the style of a given image on an increasing scale while discarding information of the global arrangement of the scene.
**Content reconstruction:** gradient descent on a white-noise image to find an image that matches the filter responses.

So let \( \overline{p} \) and \( \overline{x} \) be the original image and the image that is generated and \( P^l \) and \( F^l \) their respective feature representation in layer \( l \). We then define the squared-error loss between the two feature representations

\[
\mathcal{L}_{\text{content}}(\overline{p}, \overline{x}, l) = \frac{1}{2} \sum_{i,j} (F^l_{ij} - P^l_{ij})^2.
\]  

(1)

**Style representation:**

Gram matrix \( G^l \in \mathcal{R}^{N_l \times N_l} \), where \( G^l_{ij} \) is the inner product between the vectorised feature map \( i \) and \( j \) in layer \( l \):

\[
G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}.
\]  

(3)
Style reconstruction:

To generate a texture that matches the style of a given image (Fig 1, style reconstructions), we use gradient descent from a white noise image to find another image that matches the style representation of the original image. This is done by minimising the mean-squared distance between the entries of the Gram matrix from the original image and the Gram matrix of the image to be generated. So let $\bar{a}$ and $\bar{x}$ be the original image and the image that is generated and $A^l$ and $G^l$ their respective style representations in layer $l$. The contribution of that layer to the total loss is then

$$ E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 $$

(4)

and the total loss is

$$ \mathcal{L}_{style}(\bar{a}, \bar{x}) = \sum_{l=0}^{L} w_l E_l $$

(5)

where $w_l$ are weighting factors of the contribution of each layer to the total loss (see below for specific values of $w_l$ in our results).
Overall loss:

\[ \mathcal{L}_{\text{total}}(\tilde{p}, \tilde{a}, \tilde{x}) = \alpha \mathcal{L}_{\text{content}}(\tilde{p}, \tilde{x}) + \beta \mathcal{L}_{\text{style}}(\tilde{a}, \tilde{x}) \]
Figure 3: Detailed results for the style of the painting *Composition VII* by Wassily Kandinsky. The rows show the result of matching the style representation of increasing subsets of the CNN layers (see Methods). We find that the local image structures captured by the style representation increase in size and complexity when including style features from higher layers of the network. This can be explained by the increasing receptive field sizes and feature complexity along the network’s processing hierarchy. The columns show different relative weightings between the content and style reconstruction. The number above each column indicates the ratio $\alpha / \beta$ between the emphasis on matching the content of the photograph and the style of the artwork (see Methods).
Other Applications

- Tracking (Bazzani et. al. 2010, and many others)

- Pose estimation (Toshev et al. 2013, Jain et al., 2013, ...)

- Caption generation (Vinyals et al. 2015, Xu et al. 2015, ...)

A woman is throwing a frisbee in a park. A dog is standing on a hardwood floor. A stop sign is on a road with a mountain in the background.
Convoultional networks for music recommendation

Image from: http://benanne.github.io/2014/08/05/spotify-cnns.html
TO WRAP UP
CNNs:
summary & future directions

• Less parameters
• Allows going deeper
• High flexibility
  – In operations
  – In organization of layers
  – In the overall architecture etc.
• Future directions:
  – Understanding them better
  – Making them deeper, faster and more efficient
  – Compressing a big network into a smaller & cheaper one.
  – ...