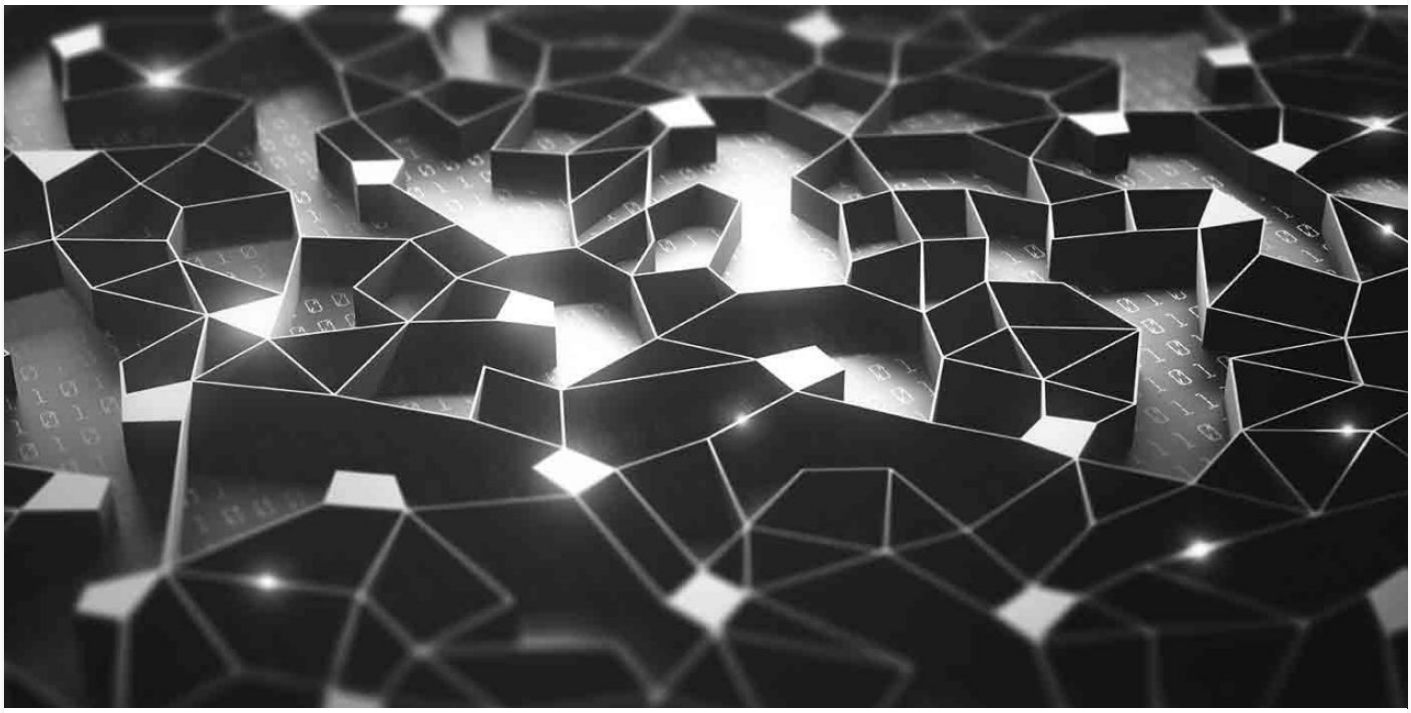
14 DESIGN PATTERNS TO IMPROVE YOUR CONVOLUTIONAL NEURAL NETWORKS

Posted by Mariya Yao | Mar 22, 2017

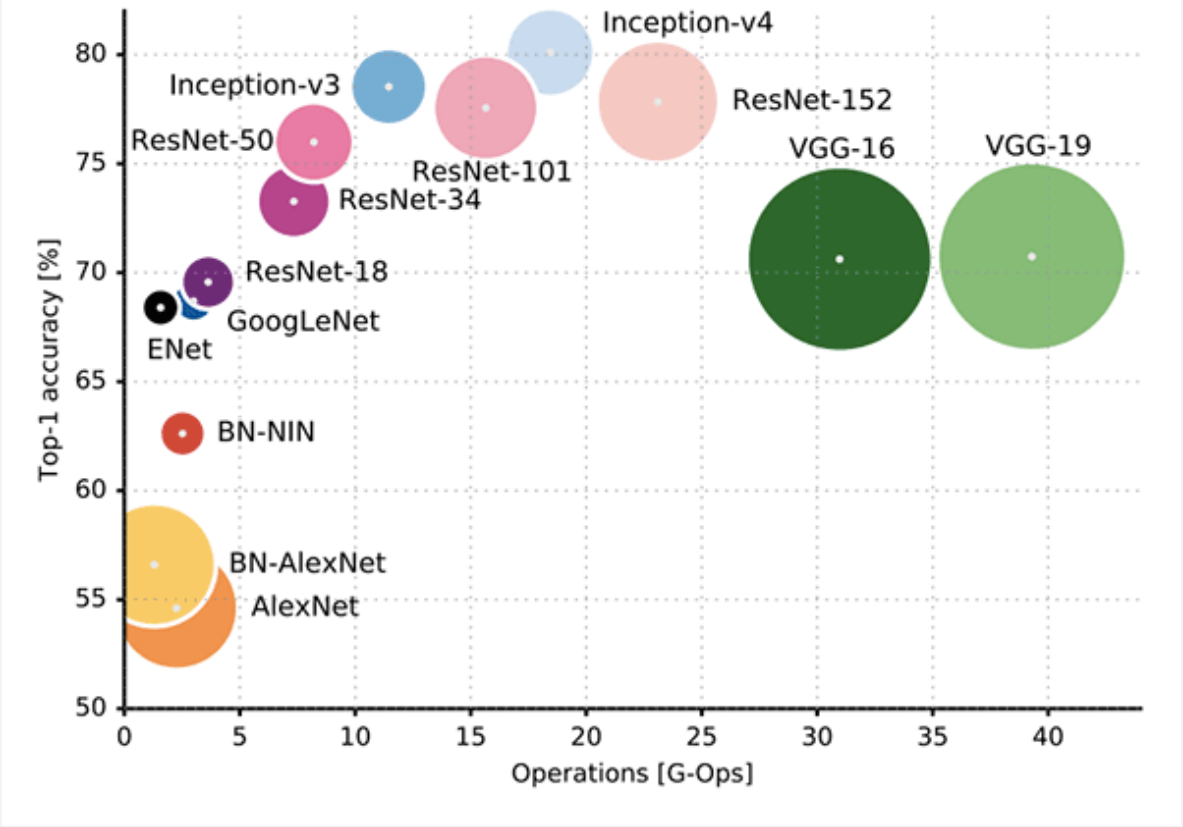


Ever since deep convolutional neural networks (CNNs) outperformed humans in image classification tasks in 2011, they have been the industry go-to standard for tasks in computer vision like image segmentation, object detection, scene labeling, tracking, text detection, and more.

Unfortunately, the art of training neural networks is not easy to master. As with previous machine learning methods, the devil is in the details, but there are so many more details to manage. What are the limitations of your data and your hardware? Should you start with AlexNet, VGG, GoogLeNet (Inception), or ResNet? There's even a ResNet in ResNet option. How many dense layers versus convolutional layers should you build? What about your activation function? Even if you decided on the popular ReLU, you then have to decide on regular ReLU, Leaky ReLU, Very Leaky ReLU, RReLU, PReLU, or the generalized version ELU.

With simple models, you might get away with leaving your learning rates at 0.001, but more complex solutions typically require more careful hyperparameter tuning. Go too small and you'll never converge on a solution. Go too big and you might blast right past it. Even adaptive learning rate approaches might be too computationally expensive depending on your hardware.

Design choices and hyperparameter settings heavily impact the training and performance of a CNN, but resources are scarce and scattered for new entrants to the field of deep learning to build an intuition for designing architectures.



Everyone knows VGG is slow and big, but you can still produce state-of-the-art results and performant business applications if you tune it correctly. Image Credit: Canziani, et al.

The primary book focused on practical tuning, Neural Networks: Tricks Of The Trade (Orr & Muller), was originally published in 2003 and updated in 2012. The hype around deep learning started when the New York Times covered the surprising win of the Merck Drug Discovery Challenge by Geoffrey Hinton's team in 2012, so the state-of-the-art research in the last few years is missing.

Luckily, several researchers, such as Leslie Smith of the U.S. Naval Research Laboratory, have published systematic studies of architectural improvements and

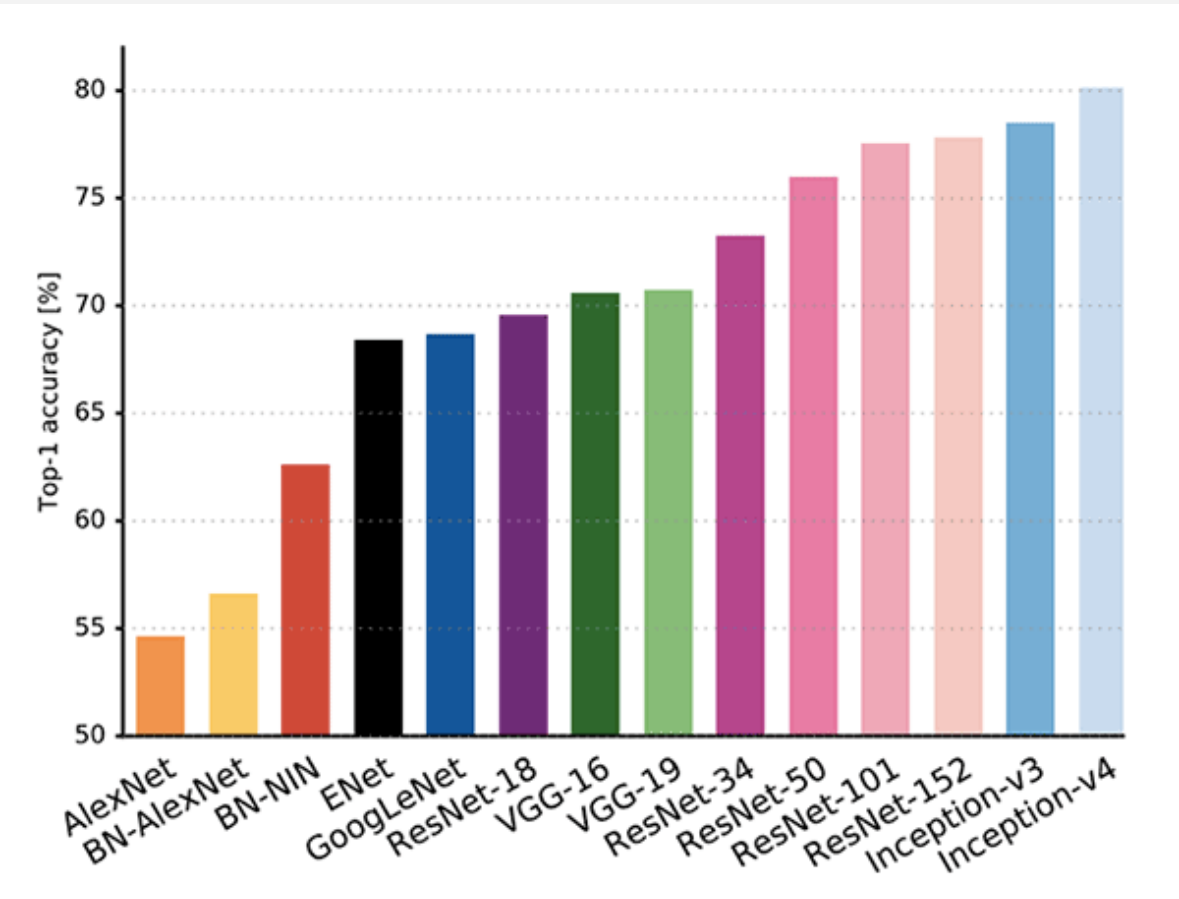
technical advances for CNNs. Here are the design patterns he has highlighted as most useful.

CNN DESIGN PATTERNS FOR IMAGE CLASSIFICATION

According to Smith, these "14 original design patterns could benefit inexperienced practitioners who seek to incorporate deep learning in various new applications." While advanced A.I. researchers can rely on intuition, experience, and targeted experimentation, this advice is a great starting point for the rest of us who don't flaunt machine learning PhDs.

1) ARCHITECTURE FOLLOWS APPLICATION

You might be wooed by shiny new models invented in fancy research labs like Google Brain or DeepMind, but many of these are either impossible or highly impractical to implement for your use case or business environment. Use the model that makes the most sense for your specific application, which may be a simple but still powerful one like VGG.



Newer architectures may be more accurate on academic benchmarks, but you should choose the architecture you understand and is best suited for your application. Image Credit: Canziani, et al

2) PROLIFERATE PATHS

Each year's winner of the ImageNet Challenge uses a deeper net than the previous year's champion. From AlexNet to Inception to Resnets, Smith and his team also observed the trend of "multiplying the number of paths through the network" and that "ResNet can be an exponential ensemble of networks with different lengths."

3) STRIVE FOR SIMPLICITY

Bigger is not necessarily better, however. In a paper by the same name, Springenberg et al demonstrate how to achieve state-of-the-art results with fewer units.

4) INCREASE SYMMETRY

Symmetry, whether architectural or biological, is considered a sign of quality and craftsmanship. Smith attributes FractalNet's elegance to the network's symmetry.

5) PYRAMID SHAPE

You are always trading off between representational power and removing redundant or useless information. CNNs universally downsample the activations and increase channels from the input layer to the final layer.

6) OVERTRAIN

Another trade-off is training accuracy vs. generalization ability. Regularization with methods like drop-out or drop-path improves generalization, which is a key advantage of neural networks. Train your network on a harder problem than your actual use case to improve generalization performance.

7) COVER THE PROBLEM SPACE

Use noise and data augmentation – such as randomized rotations, crops, and image manipulations – in order to expand your training data and improve generalization

8) INCREMENTAL FEATURE CONSTRUCTION

As architectures become more successful, they further simplify each layer's "job". In very deep neural networks, each layer only incrementally modifies the input. In ResNets, the output of a layer is likely to be similar to the input, which means adding the two is incremental. In practice, use short skip lengths in ResNet.

9) NORMALIZE LAYER INPUTS

Normalization is another approach to making a layer's work easier and is shown in practice to improve training and accuracy. The inventors of batch normalization believe the cause lies in handling internal covariate shift, but Smith believes that normalization "puts all the layer's input samples on ore equal footing (analogous to a units conversion scaling), which allows back-propagation to train more effectively."

10) INPUT TRANSITION

In Wide ResNets, research showed that performance improves with the number of channels, but you trade cost vs. accuracy. AlexNet, VGG, Inception, and ResNets all do this in the first layer to enable the input data to be examined in many ways.

11) AVAILABLE RESOURCES GUIDE LAYER WIDTHS

The number of outputs to choose is not obvious, however, and depends on your hardware capabilities and desired accuracy.

12) SUMMATION JOINING

Summation is a popular way to combine branches. In ResNets, using sums as a joining mechanism enables each branch to compute residuals vs. the whole approximation. If the input skip connection is always present, then summation causes the layers to learn the corrective terms (i.e. difference from the input). In networks with several branches where any branch can be dropped (i.e. FractalNet), you should use means to keep the output smooth.

13) DOWN-SAMPLING TRANSITION

When pooling, use concatenation joining for increasing the number of outputs. When using a stride greater than 1, this simultaneous handles joining and increases the number of channels.

14) MAXOUT FOR COMPETITION

Maxout is used in locally competitive networks where you choose only one of the activations. Using sums and means includes all activations, so the difference is that max out chooses only one "winner". One clear use case for Maxout is when each branch has different sized kernels and Maxout is able to incorporate scale invariance.

TIPS & TRICKS

In addition to these design patterns, several recent tips and tricks have emerged for reducing architectural complexity and training time and working with noisy labels.

1) USE PRE-TRAINED NETWORKS WITH FINETUNING

"If your visual data is similar to ImageNet's, using a pretrained network helps you learn faster," explains Mike Tung, CEO of machine learning company Diffbot. Lower levels of a convolutional neural net can typically be reused since they detect common patterns like lines and edges. Replace the classification layer with your own and finetune the last few layers to your specific data.

2) USE FREEZE-DROP-PATH

Drop-path randomly removes branches during an iteration of training. Smith tested an opposite method called freeze-path, where the branches weights are frozen and untrainable rather than entirely removed. The network should attain greater accuracy since the next branch contains more layers than the previous branch and the corrective term is easier to approximate.

3) USE CYCLICAL LEARNING RATES

Experimenting with learning rates consumes time and exposes you to errors. Adaptive learning rates can be computationally expensive, but cyclical learning rates are not. With a CLR, you set minimum and maximum boundaries and vary the learning rate between them. Smith even offers a semi-automated method for computing your max and min in his paper on the topic.

4) USE BOOTSTRAPPING WITH NOISY LABELS

In practice, most of your data will be messy, where labels are subjective or missing and objects might not be localized. Reed et al describes a method to inject consistency into a network's prediction objective. Intuitively this works by enabling the network to make use of known representations of the world (implied in parameters) to filter out incoming data that may have an inconsistent training label and clean up that label while training.

5) USE ELUS NOT RELUS, ESPECIALLY WITH MAXOUT

ELUs are smoother versions of ReLUs that speed up convergence and classification accuracy. Unlike ReLUs, ELUs have negative values which allows them to "push mean unit activations closer to zero like batch normalization but with lower computational complexity" according to research. They're particularly effective if you're using Maxout with fully connected layers.

6) EXPERIMENT WITH WEIGHT INITIALIZATION

Much of tuning a neural network is art, not science, and new best practices are discovered every day. So, you might as well experiment and see what works for you. Andrej Karpathy has an introductory overview of various approaches to weight initialization, which can dramatically affect your speed to convergence or whether you converge at all. Just be sure not to initialize to all zeros!

RECOMMENDED READING

An Analysis of Deep Neural Network Models for Practical Applications
Alfredo Canziani, Adam Paszke, Eugenio Culurciello
Revised February 23rd, 2017

Cyclical Learning Rates for Training Neural Networks
Leslie N. Smith
Revised Dec 29th, 2016

Deep Convolutional Neural Network Design Patterns
Leslie N. Smith, Nicholay Topin
Revised November 14th, 2016

Systematic evaluation of CNN advances on the ImageNet
Dmytro Mishkin, Nikolay Sergievskiy, Jiri Matas
Revised June 13, 2016

Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)
Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter
Revised February 2016

Training Deep Neural Networks on Noisy Labels with Bootstrapping
Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, et al
Revised April 15th, 2015

Special thanks to Jeremy Howard and Rachel Thomas of fast.ai for recommending these papers and Mike Tung of Diffbot for sanity checking technical details and adding commentary.

Did we miss any important or useful papers for designing and optimizing your CNNs? Let us know in the comments below.

ABOUT THE AUTHOR

Mariya Yao

Mariya is Chief Technology Officer and Head of Product at Metamaven, which grows revenues for Fortune 500 enterprises using data science, machine learning, and automation. She "translates" arcane technical concepts into actionable business advice for executives and designs lovable products people actually want to use. Follow her on Twitter at @thinkmariya to raise your AI IQ.

1 COMMENT

InYan on March 26, 2017 at 7:32 pm

Very nice article. If you interested in visiting AI & chatbot conferences in 2017 please take a look at this list — https://medium.com/standuply/best-ai-and-chatbot-conferences-in-2017-55f020acb782

REPLY