Course Notes: Deep Learning for Visual Computing

Peter Wonka

August 30, 2021

Contents

1 Network Analysis

1.1 What is the representational power of a neural network?

- A fully-connected network defines a family of functions parametrized by the weights
- What is the representational power of this family of functions?
- Are there functions that cannot be modeled with a neural network?

1.1.1 Representational Power of a Linear Network

- Assume a network consisting of only linear layers:
  - $\mathbf{x_2} = \boldsymbol{W}_1\mathbf{x_{in}}$
  - $\mathbf{x_3} = \boldsymbol{W}_2\mathbf{x_2}$
  - ...
  - $\mathbf{x_{out}} = \boldsymbol{W}_L\mathbf{x_L}$
- $\mathbf{x_{out}} = \boldsymbol{W}_L...\boldsymbol{W}_2\boldsymbol{W}_1\mathbf{x_{in}}$
- Combination of many linear layers is a single linear layer

1.1.2 Representational Power

- Networks with at least one hidden layer are **universal function approximators**
  - as always certain restrictions and conditions apply
- Graduate school version:
  - Literature: Cybenko, **Approximation by superpositions of a sigmoidal function**, Mathematics of Control, Signals and Systems 1989
  - "We show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity."
  - Literature: Hornik et al., **Multilayer feedforward networks are universal approximators**, 1989
- High school version
  - Michael Nielsen: youtube 6 min

- Michael Nielsen: online book with interactive graphs
- Given any continuous function $f(x)$ and some $\epsilon > 0$, there exists a Neural Network $g(x;\theta)$ with one hidden layer with a reasonable choice of non-linearity (e.g. sigmoid) such that $\forall x, |f(x) - g(x)| < \epsilon$

1.1.3 Representational Power Continued

- proof does not create reasonable networks in practice
- The derivatives of the feedforward network can also approximate the derivatives of the function arbitrarily well
  - Literature: Hornik et al., Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, 1990
- Original proofs were for a class of activation functions like sigmoid
  - Extension to other functions, e.g. ReLU
  - Literature: Leshno et al., Multilayer feedforwardnetworks with a nonpolynomial activation function can approximate any function, 1993
- How big does the single layer need to be?
  - Unfortunately, in the worst case, an exponential number of hidden units (one hidden unit corresponding to each input configuration that needs to be distinguished)

may be required

1.1.4 Deeper Networks Help

- Literature: Montufar et al., On the number of linear regions of deep neural networks, NIPS 2014
- Piecewise linear networks are obtained, e.g., from rectifier non-linearities or maxout units
- Piecewise linear networks can represent functions with a number of regions that is exponential in the depth of the network
- The number of linear regions carved out by a deep rectifier network is
  - d inputs
  - depth / layers l
  - n units per hidden layer

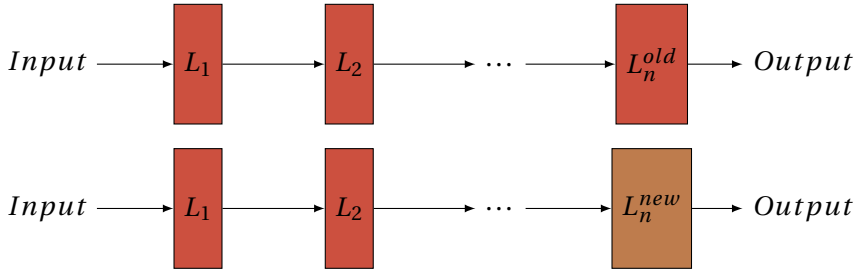
$$O\left(\binom{n}{d}^{d(l-1)} n^d\right) \tag{1.1}$$

1.1.5 Further Reading

- Deep Learning book, Bengio, Goodfellow, Courville, Chapter 6.4.

1.2 Importance of Transfer Learning

1.2.1 Setup



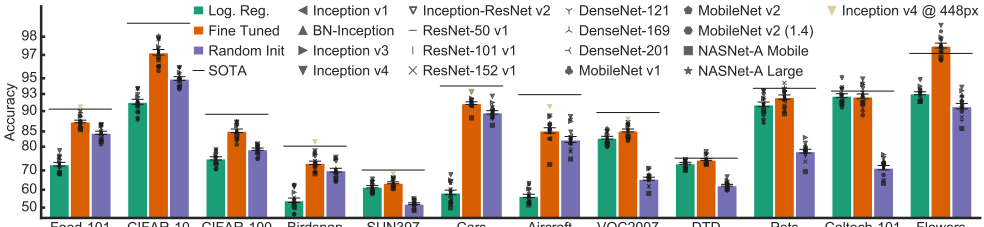
- Literature: Kornblith et al., Do Better ImageNet Models Transfer Better?, arXiv 2018, CVPR 2019
- Compare the performance of 16 classification networks on 12 image classification datasets
- Compare different versions:
  - **Logistic Regression**:

- Initialize with pre-trained ImageNet weights
- Cut off the last classification layer
- Add new classification layer with random weights
- Train **last layer** weights / keep all earlier weights fixed

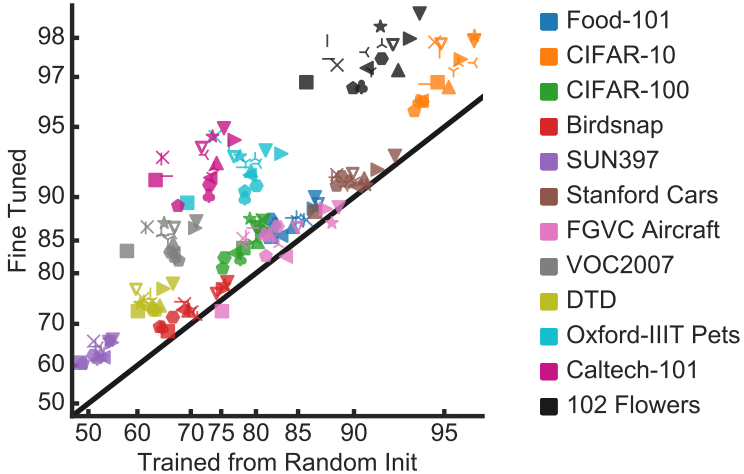
- **Fine Tuned**:
  - Initialize with pre-trained ImageNet weights
  - Cut off the last classification layer
  - Add new classification layer with random weights
  - Train **all layers** (no weights fixed)
- **Random Init** (no transfer)
  - Cut off the last classification layer
  - Add new classification layer
  - Initialize all weights randomly

- Train all layers

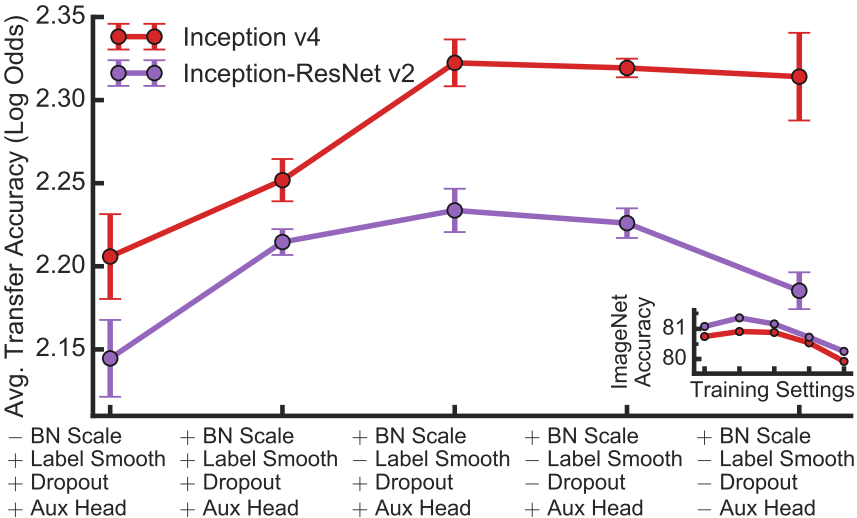
1.2.2 Pretraining is Important



- Classification performance:
  - Generally: Fine Tuning > Random Init > Logistic Regression
- Time:
  - Logistic Regression << Fine Tuning < Random Init
- Different view of the results


Fine Tuned
98
97
95
90
85
80
70
60
50
50
60
70
75
80
85
90
95
Trained from Random Init
Food-101
CIFAR-10
CIFAR-100
Birdsnap
SUN397
Stanford Cars
FGVC Aircraft
VOC2007
DTD
Oxford-IIIT Pets
Caltech-101
102 Flowers

1.2.3 Regularization can be beneficial or harmful



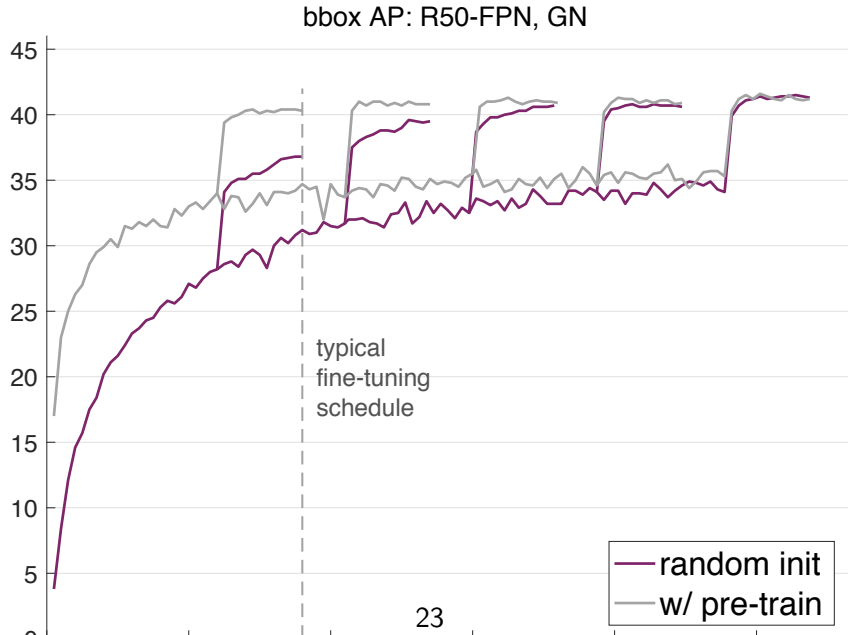
- Comparing fine-tuning for pretrained networks with and without regularization techniques

1.2.4 Conclusions

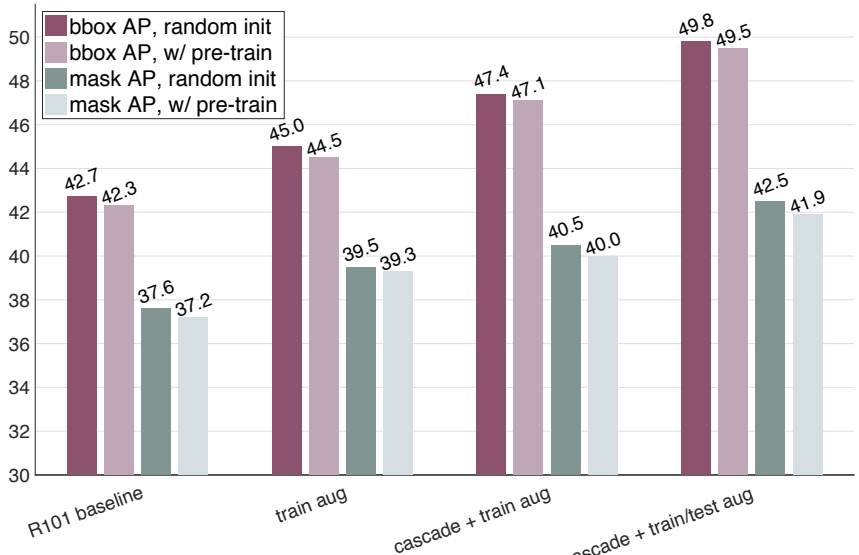
- ImageNet accuracy predicts performance of logistic regression on fixed features, but regularization settings matter
- **ImageNet accuracy predicts fine-tuning performance**
- **ImageNet accuracy predicts performance of networks trained from random initialization**
- ImageNet pretraining does not necessarily improve accuracy on fine-grained tasks
- ImageNet pretraining accelerates convergence
- Accuracy benefits of ImageNet pretraining fade quickly with dataset size

1.3 Rethinking ImageNet Pretraining

- Literature: He et al., Rethinking ImageNet Pre-training, arXiv 2018, ICCV 2019
- ImageNet pretraining might only provide accelerated training, no accuracy improvement for object detection


bbox AP: R50-FPN, GN
45
40
35
30
25
20
15
10
5
typical
fine-tuning
schedule
random init
w/ pre-train
23

- The jumps in the figure are learning rate decreases


bbox AP, random init
bbox AP, w/ pre-train
mask AP, random init
mask AP, w/ pre-train
50
48
46
44
42
40
38
36
34
32
30
42.7
42.3
37.6
37.2
45.0
44.5
39.5
39.3
47.4
47.1
40.5
40.0
49.8
49.5
42.5
41.9
R101 baseline
train aug
cascade + train aug
cascade + train/test aug

- Training converges to better results without pre-training

1.4 How big to make the network?

- Choromanska et al., The Loss Surfaces of Multilayer Networks, arXiv 2014, JMLR 2015

1.5 Deep Networks can Learn Random Labels

- Literature: Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, Oriol Vinyals, **Understanding deep learning requires rethinking generalization**, ICLR 2017 (Best paper award)
- Experiment:
  - assign random class labels to images
  - can the network learn random class labels?
  - what is the generalization error of random class labels?

dog

banana

banana

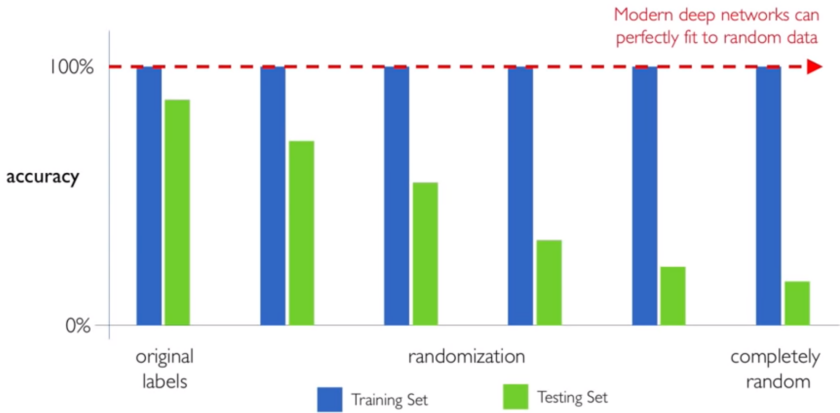
dog

dog

tree

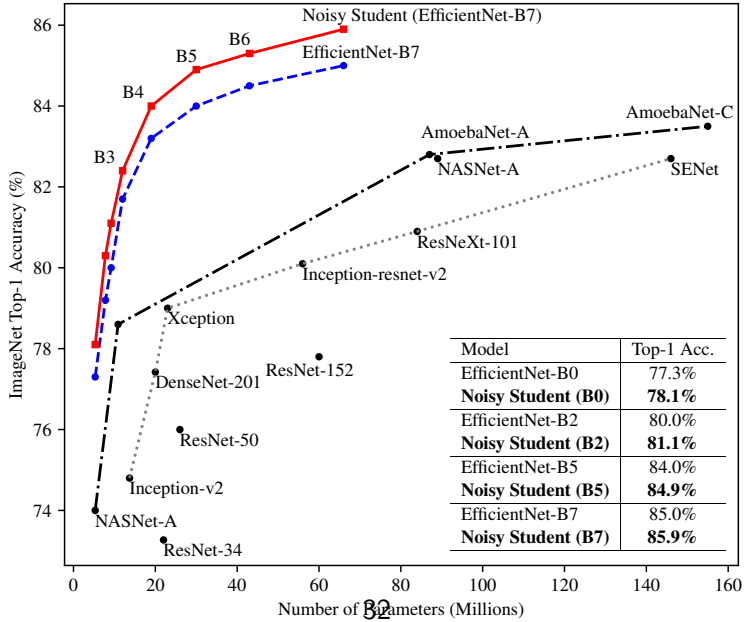
tree

dog


Modern deep networks can perfectly fit to random data
100%
accuracy
0%
original labels
randomization
completely random
Training Set
Testing Set

1.6 Teacher Student Training

- Literature: Xie et al., Self-training with Noisy Student improves ImageNet classification, arXiv November 2019
- Improved Training with EfficientNet



Model	Top-1 Acc.
EfficientNet-B0	77.3%
Noisy Student (B0)	**78.1%**
EfficientNet-B2	80.0%
Noisy Student (B2)	**81.1%**
EfficientNet-B5	84.0%
Noisy Student (B5)	**84.9%**
EfficientNet-B7	85.0%
Noisy Student (B7)	**85.9%**

- Term: **pseudo label** is a label assigned by a trained network
- Hard vs. Soft Labels
  - Hard labels are 1 for the correct class and 0 for other classes
  - Soft labels are softmax probabilities (work better)
- Main Ideas:
  - self-training (a method in semi-supervised learning)
  - add many sources of variation (noise) to the student
  - no noise in the teacher when generating pseudo labels
- Sources of noise:
  - dropout
  - stochastic depth
  - data augmentation
- Data balancing for pseudo labels

- duplicate images in classes with not enough images
- take highest confidence images for classes with too many images

Algorithm 1: Noisy Student method

Require: Labeled images $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ and unlabeled images $\{\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_m\}$.

1: Learn teacher model θ_* which minimizes the cross entropy loss on labeled images

$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f^{noised}(x_i, \theta))$$

2: Use an unnoised teacher model to generate soft or hard pseudo labels for unlabeled images

$$\tilde{y}_i = f(\tilde{x}_i, \theta_*), \forall i = 1, \cdots, m$$

3: Learn student model θ'_* which minimizes the cross entropy loss on labeled images and unlabeled images with noise added to the student model

$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f^{noised}(x_i, \theta')) + \frac{1}{m}\sum_{i=1}^{m} \ell(\tilde{y}_i, f^{noised}(\tilde{x}_i, \theta'))$$

4: Iterative training: Use the student as a teacher and go back to step 2.

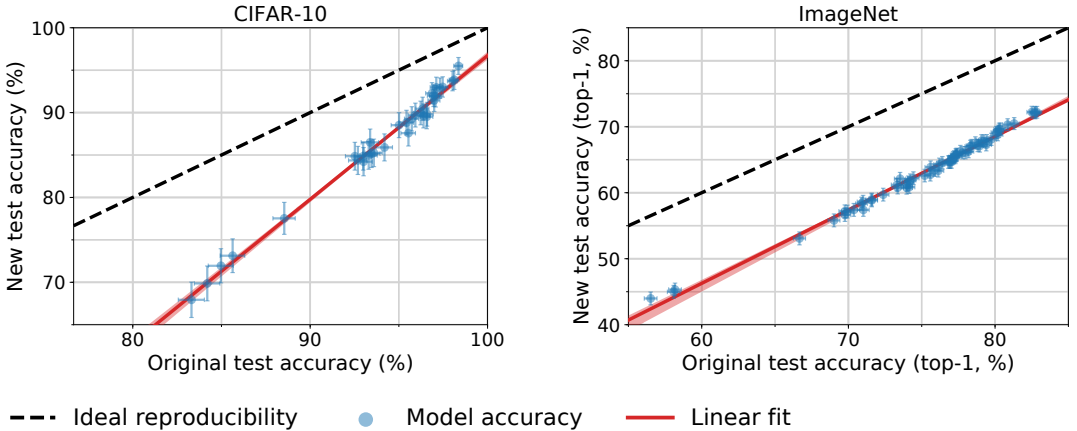
1.7 Do ImageNet Classifiers Generalize to ImageNet?

- Literature: Recht et al., Do ImageNet Classifiers Generalize to ImageNet?
- Are more recent networks better or do they just overfit the training data?
- What is the effect of using the same test data over and over again?

1.7.1 Test Setup

- Create a new test set for **CIFAR-10** and **ImageNet**
- Follow the original protocol for dataset creation
- Evaluate networks trained on CIFAR-10 and ImageNet on the new test sets

1.7.2 Main Result



- Model accuracy on the **original test sets** vs. **new test sets**
- Each data point corresponds to one model (shown with 95% Clopper-Pearson confidence intervals)
- Two main conclusions:
  - Significant drop in accuracy from the original test sets to the new test sets
  - Model accuracies closely follow a linear function with slope **greater than 1** (1.7 for CIFAR-10 and 1.1 for ImageNet)
    - Every percentage point of progress on the original test set translates into more than one percentage point on the new test set.

1.7.3 Analysis

- Possible causes for the difference in performance:
  - **Generalization Gap**: new data has harder samples due to **random sampling**
    - Difference in performance is too large
  - **Distribution Gap**: new data has harder samples due to **systematic differences**
    - e.g. cameras change over time, data generation protocol slightly different, ...
    - authors conjecture this is the main reason
  - **Adaptivity Gap**: networks are adapted to the test set
    - e.g. by directly training on the test set, tuning hyperparameters on the test set, architecture choices because of test set, ...
    - not likely because of the second conclusion above