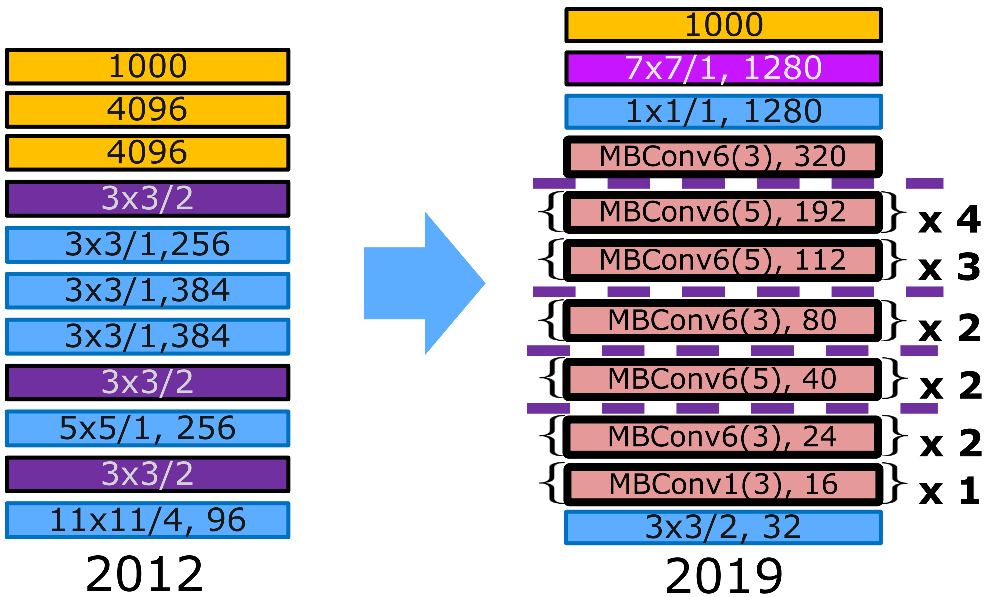
Photogrammetry & Robotics Lab

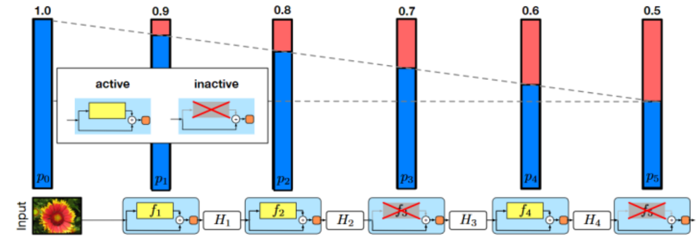
Machine Learning for Robotics and Computer Vision Tutorial

CNNs in Practice

Jens Behley

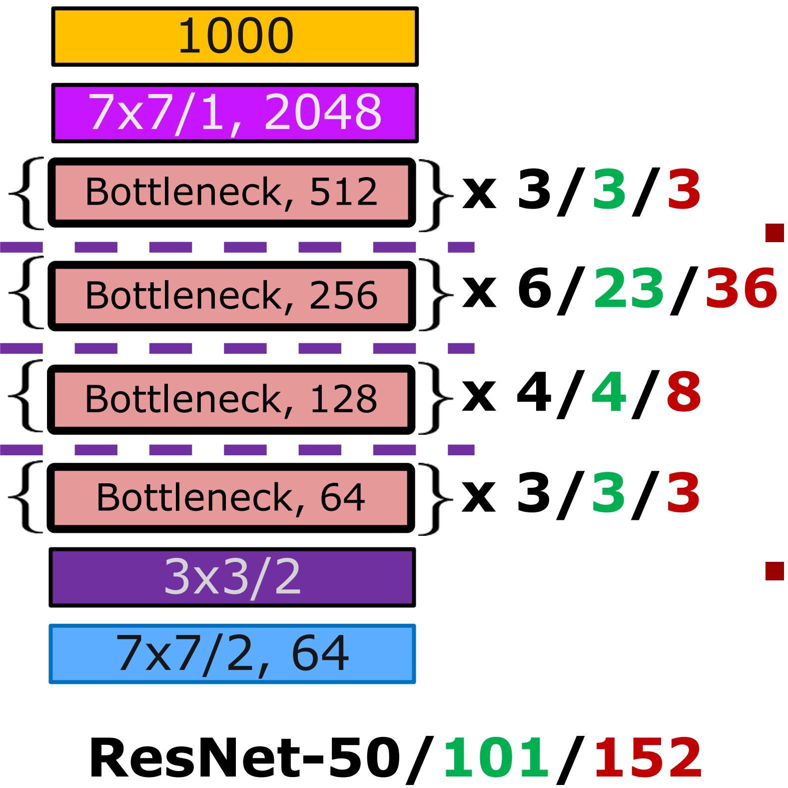
Last Lecture





- Popular and significant architectures and changes
  - global avg pooling → skip connection → efficiency
  - VGG → GoogleLeNet → ResNet → MobileNetV2 → EfficientNet
- Looked at common ways to close the gap between training and test performance (generalization gap)

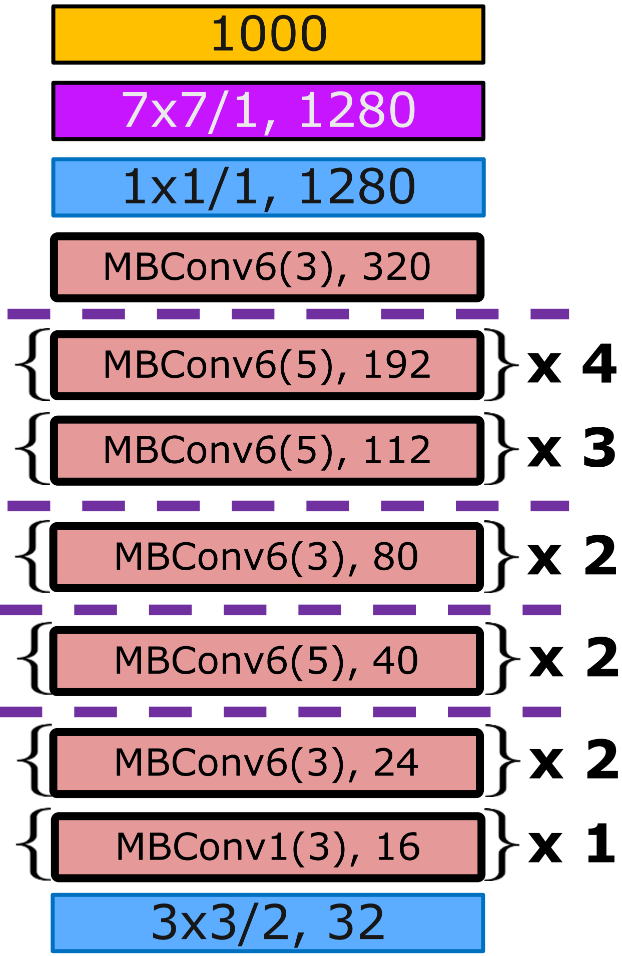
Recap: ResNet (2015)



- 18/34/50/101/152 learnable layers
- Downsampling (– –) via strided (s=2) convolution in first convolution of 2nd, 3rd, 4th stage
- ResNet-152 reaches 3.6% Top 5-error by ensemble of 6 models

[He, 2015]

Recap: EfficientNet (2019)



EfficientNet-B0

[Tan, 2019]

- NAS with objective of accuracy and computational efficiency (FLOPS)
- Structural very similar to MobileNetV2
  - Different number of channels
  - Convs with 3x3 and 5x5 kernels
- MBConv6(k) is inverted bottleneck with t=6 and k x k depth-wise separable convolutions

Model	Top-1	#Params
EfficientNet-B0	**77.1**	5.3M
ResNet-50	76.0	26 M
MobileNetV2	72.0	**3.4M**

Which architecture to choose?

- **ResNet-50** is a good baseline model that is often used and shows good performance

- Nowadays, **EfficientNets** are efficient alternative and current research focuses on more on training efficiency

- Vibrant research field → novel insights
- Improved training strategies shows promising improvement even for "old" ResNets

ResNet in PyTorch

```
import torchvision.models as models


# initialize resnet with FC layer
outputting 10 scores instead of 1000

resnet50 = models.resnet50(num_classes=10)
```

- In torchvision are already many models implemented
- Rich set of options, see documentation.

- Can be used as building block (replace fc with indentity module if fully-connected layer not needed)

Best Practices

- Some advice on how to approach a new problem (data, research problem, …)

- If common vision task: get a quick baseline working
→ pre-trained networks, existing approaches, e.g., torchvision has off-the-shelf implementations for classification, detection, semantic/instance segmentation & tutorials
- Insights about the task; this is the thing that you want to "beat"

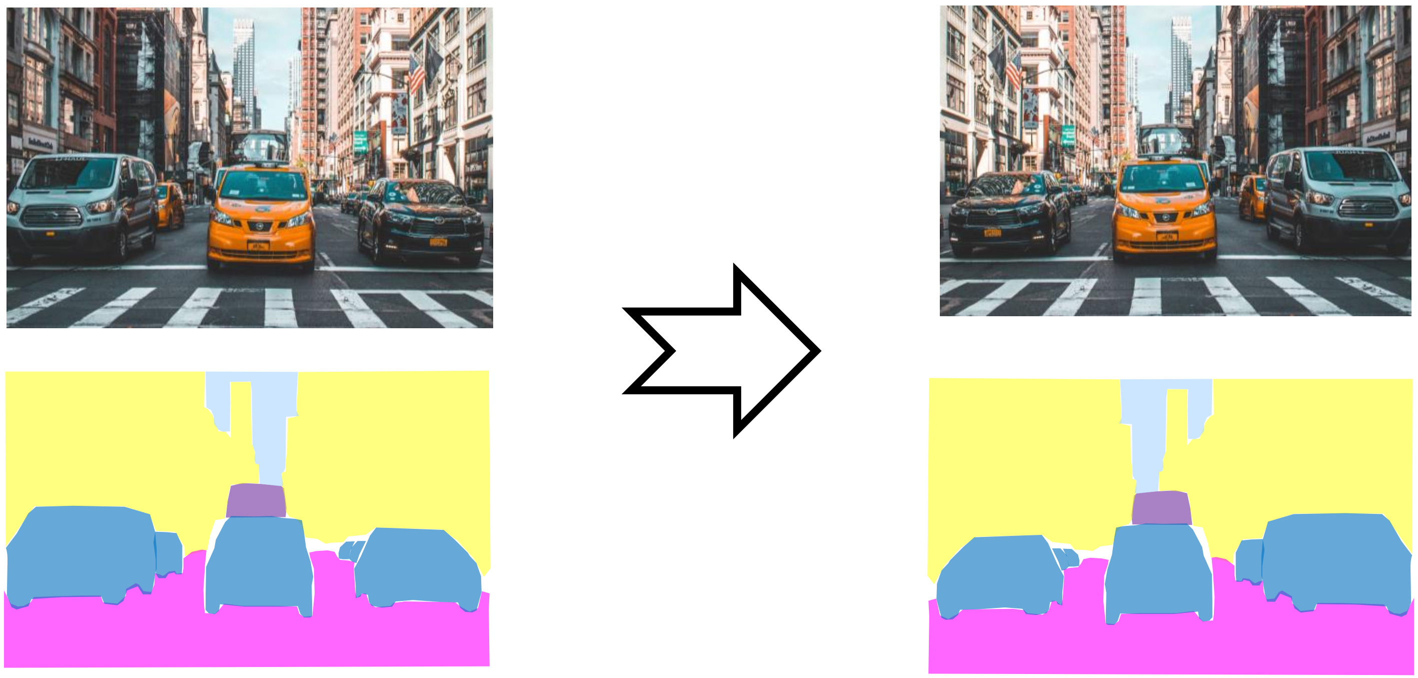
Be paranoid!

- Check everything, plot/visualize images
- Often subtle errors in parts that you think are easy, you did 1000 times
  → Check especially the easy parts

- Beauty, but also danger of Machine Learning: Even if the objective does not make sense or is erroneous, gradient descent will still optimize this.

- **Beware:** CNNs can still do a good job, even though there are errors in the pipeline or data

Example: Data augmentation in semantic segmentation



- Horizontal flip of image, must be also applied to pixel-wise labels
- A CNN can still produce reasonable results with only flipped images, but performance will drop!

Again: Getting the Data Right

- You will run many, many experiments, variations of your model, etc.
- Have a **validation set** ready that is large enough and can be used to assess your progress

- You want to know if you are making progress; if validation set changes all the time, you get “noisy” feedback and cannot compare to earlier steps

General workflow

- Common receipt
  1. First get **simple pipeline** working with small number of images
     → general problems or issues will not be magically solved by putting more data in!

General workflow

- Common receipt
  1. First get **simple pipeline** working with small number of images
     → general problems or issues will not be magically solved by putting more data in!
  2. **Monitor learning progress** via loss curves & metrics
     → Use tensorboard to visualize logs

General workflow

- Common receipt
  1. First get **simple pipeline** working with small number of images
     → general problems or issues will not be magically solved by putting more data in!
  2. **Monitor learning progress** via loss curves & metrics
     → Use tensorboard to visualize logs
  3. **Improve model** by analyzing behavior on train/validation or dev set
     → adding additional things (maybe going **back to step 1**)

Step 1: Getting Started

- Always start with a simple pipeline and only some examples
- First experiments with only little data
- Does everything work as expected

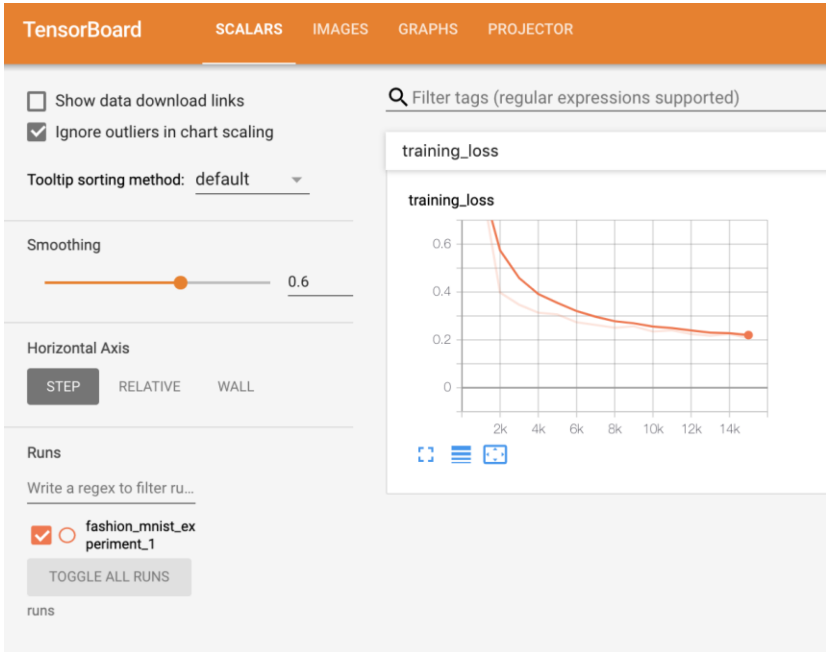
Step 1: General Receipt

- Receipt:

1. No data augmentation, regularization, dropout, etc. (added later)
2. Single batch of few images (5-10 minibatches) → Network should be able to reach 100%
3. Check if possible to overfit on small data
4. Determine learning rate on all data on small number of iterations (LR: 0.1,0.01, 0.001, …)
5. Determine better learning rate/weight decay (1-5 epochs)
6. Train longer → Monitor progress

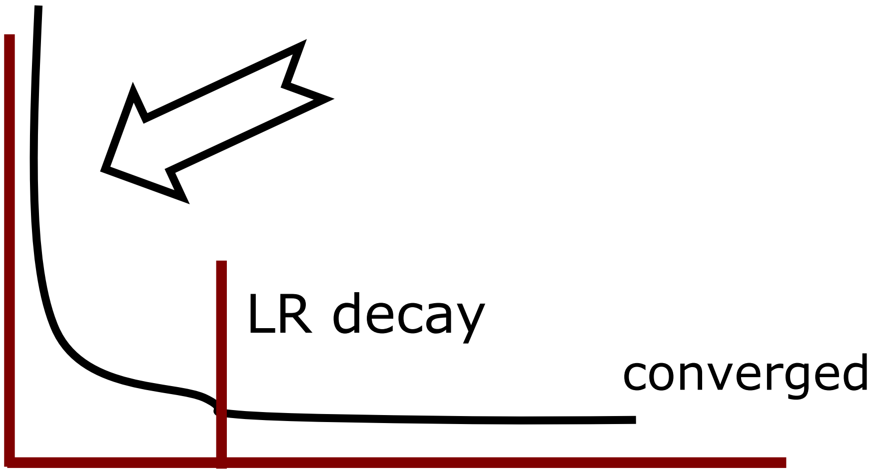
Inspired by Karpathy's & Johnson's advice

Step 2: Babysitting



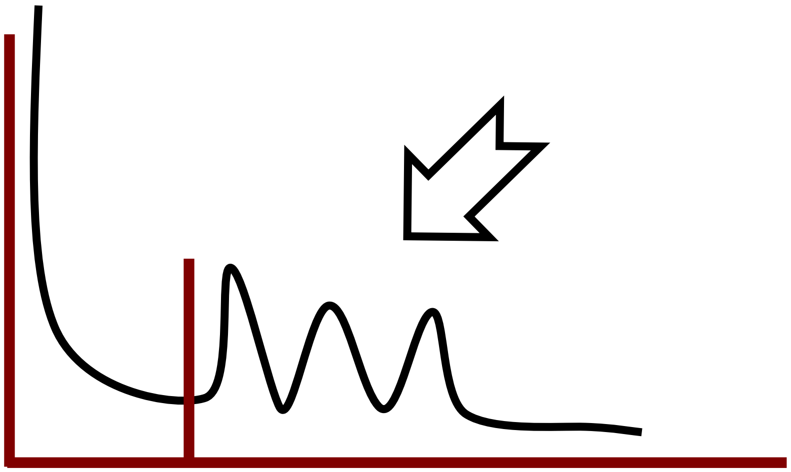
- Use tensorboard to visualize loss (train, validation) & metrics of interest
- Visualizes logs that are produced while training
- See tutorial Pytorch’s tutorial on tensorboard

Learning Curves



- Optimal curves look like decreasing fast, no larger "upticks"

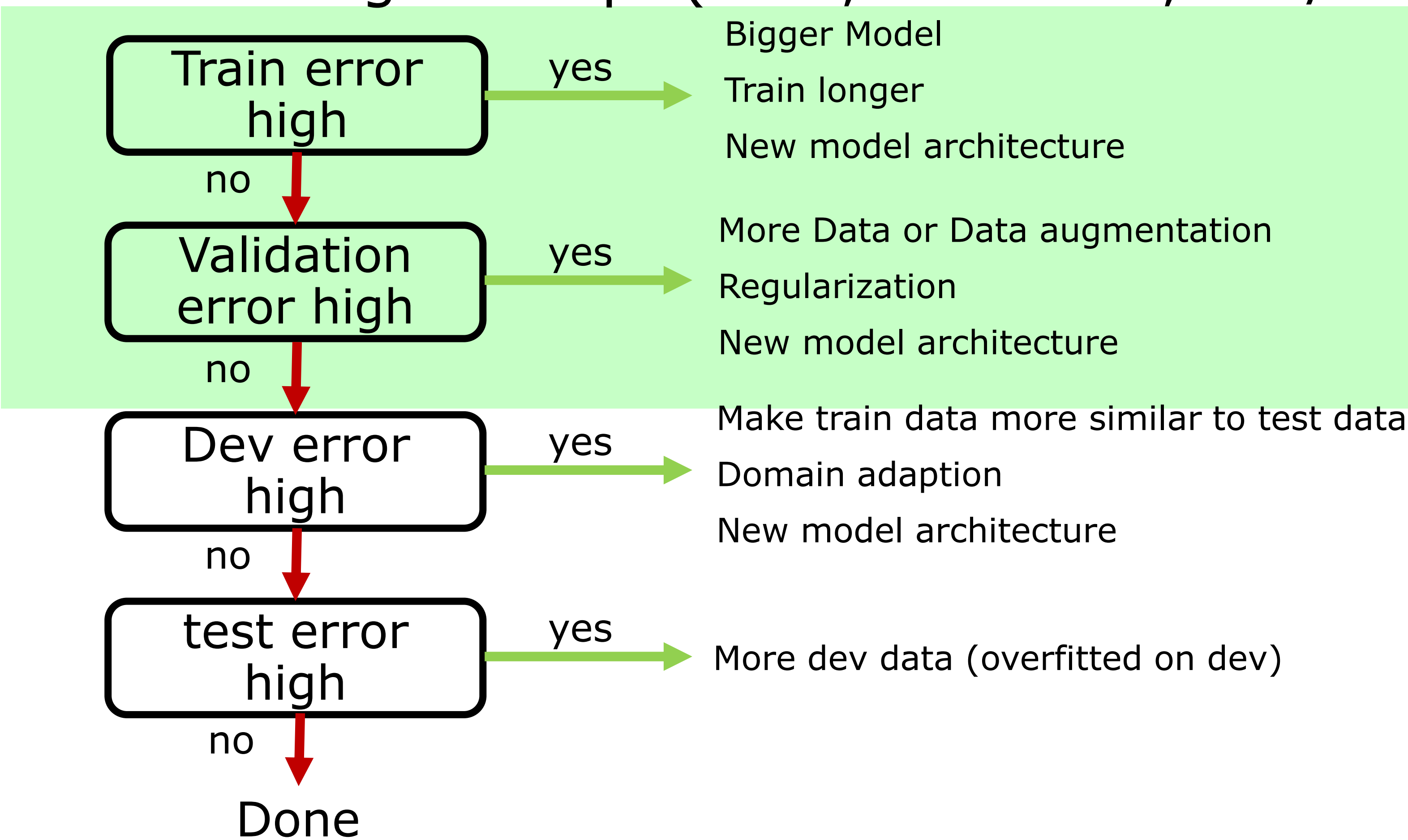
Learning Curves – LR too high



- Spikes are never a good sign, even if loss decreases afterwards
- Usually an indicator that LR is to high: decay LR before

Step 3: Improve Model

- Asses performance and improve model
- Andrew Ng’s receipt (train, validation, dev/test set)



[Machine Learning Yearning]

Step 3: Trying Things

- Make always one change, like adding data augmentation, adding layers, etc.
- Here, the fixed validation set will help to figure out if you are on the right track

- Get inspiration from similar work. Look at latest papers on ImageNet classification
- There is no single way or algorithm that you

Additional Reading

- Other resources on best practices & strategies:

1. Justin Johnson’s Notes in Lecture PDF
2. Andre Karpathy’s Notes PDF
3. Andrew Ng. Machine Learning Yearning PDF

See you next week!