LUND UNIVERSITY

FMAN-45: Machine Learning

Lecture 9:
Regularization and Optimization
Advanced Deep Learning Topics

David Nilsson, Cristian Sminchisescu

Training a Neural Network

Highly non-convex and extremely high-dimensional optimization problem.

Often millions of parameters, so second-order methods are out of the question. We have to resort to only using the gradient.

For deep networks, the training often takes several days using one or more GPUs.

There is not much theory on how to train neural networks. It is mostly various tricks and design patterns that have been found to be useful in practice.

A commonly used trick is to introduce randomness into the training and to average it out during testing.

Optimization

Machine learning heavily relies on optimization. Parameter values are optimized to make the predictions close to the ground truth.

The generic Empirical Risk Minimization problem: $\min_w \mathbb{E}_{x,y\sim p_{data}(x,y)}[L(f_w(x), y)] \approx \min_w \frac{1}{n}\sum_{i=1}^{n} L(f_w(x_i), y_i)$

We have some measure L of how well the prediction $f_w(x)$ match the ground truth y.

A low value of the objective function means that the predictions are (mostly) correct.

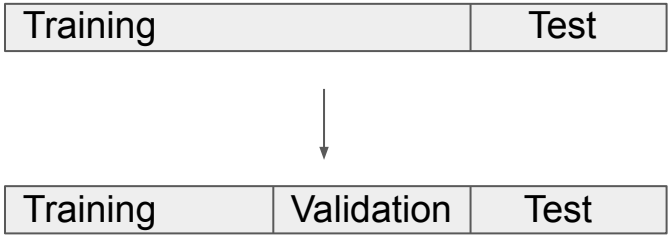
The parameters w are then used for the deployed model.

Validation Set

If we use the test set too much it becomes less and less reliables as a measure of generalization ability. If the models are tested too frequently on the test set, then the test set is in fact training data.

For many large public datasets, the test set ground truth is withheld and you can only test, say, every 48th hour, by submitting to a server.

Split the training set into training data and validation data. Use the training data for optimization and the validation data for performance assessment.

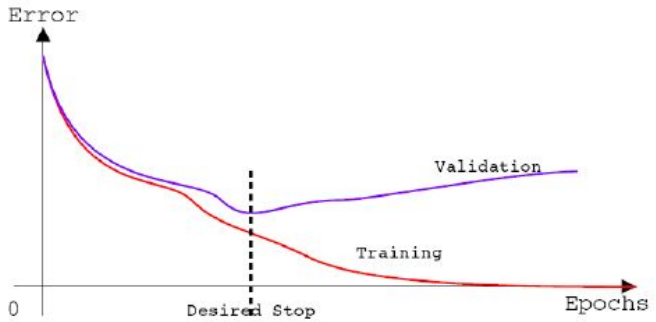


Early Stopping

Need a validation set to assess the progress of training.

Test againsts a validation set frequently during training and select the model parameters from where they were when the validation error was the lowest.

Note that the models starts to overfit the training data when the training error decreases and the validation error increases.



Early Stopping

The training and test sets approximate the true data distribution.

What we are after is good performance on the true data distribution.

When a model is deployed in a real application, the performance on the true distribution is what matters.

$$\min_w \mathbb{E}_{x,y \sim p_{data}(x,y)}[L(f_w(x), y)] \approx \min_w \frac{1}{n} \sum_{i=1}^{n} L(f_w(x_i), y_i)$$

Stochastic Gradient Descent with Momentum

When training neural networks, we always use a mini-batch and the gradient estimate can be noisy.

Compute a moving average of the gradient estimation.

The most widely used optimization method for training deep neural network, despite its simplicity.

$$
\begin{aligned}
g_t &= \nabla f(\theta_{t-1}) \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
\theta_t &= \theta_{t-1} - \alpha m_t
\end{aligned}
$$

ADAM

Compute moving averages of both the gradient and the square of the gradient (element-wise).

Works very well in practice, especially for recurrent neural networks.

Kind of constraints the magnitude of the parameter updates to be close to α, on average, for all parameters.

$$
\begin{aligned}
g_t &= \nabla f(\theta_{t-1}) \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= m_t / (1-\beta_1^t) \\
\hat{v}_t &= v_t / (1-\beta_2^t) \\
\theta_t &= \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)
\end{aligned}
$$

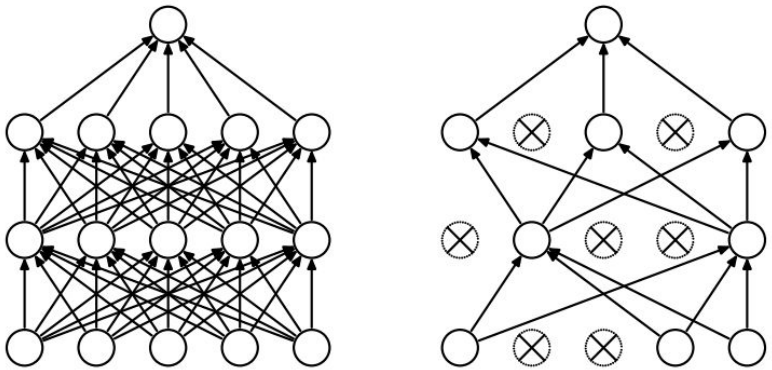
Dropout Layer

At training, randomly set feature activations to 0 with probability p and multiply the ones that remain with 1/p to keep the expected value. At test time, use the layer as normal without dropout.

Effectively an ensemble of exponentially many subnetworks. At test time all subnetworks are averaged.

Forces the features to be diverse and reduces overfitting. We can’t rely on a single very good feature.

Typically used in the very last fully connected layers in a network.



(a) Standard Neural Net (b) After applying dropout.

Srivastava et al. JMLR 2014

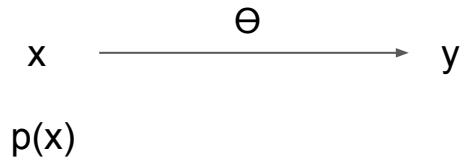
Batch Normalization Layer

Suppose that we have a layer (e.g. convolutional) from x to y. The layer has parameters Θ and maps x to y.

During training, the distribution of x changes because the earlier layers are updated. The distribution p(x) is non-stationary. The mapping x->y has to not only learn new things but also to keep up with the distribution changes in x.

Covariate shift = Change in the input distribution.

Batch normalization is a technique used to reduce the covariate shift.



Batch Normalization Layer

We reduce the scaling aspect of the distribution changes by normalizing each layer using mini-batch statistics. The output y will have mean β and standard deviation γ (both trainable).

For modern deep networks, a conv layer is always followed by batch normalization (per channel) and relu.

During training we compute moving averages of the means and standard deviations that we later use in testing.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

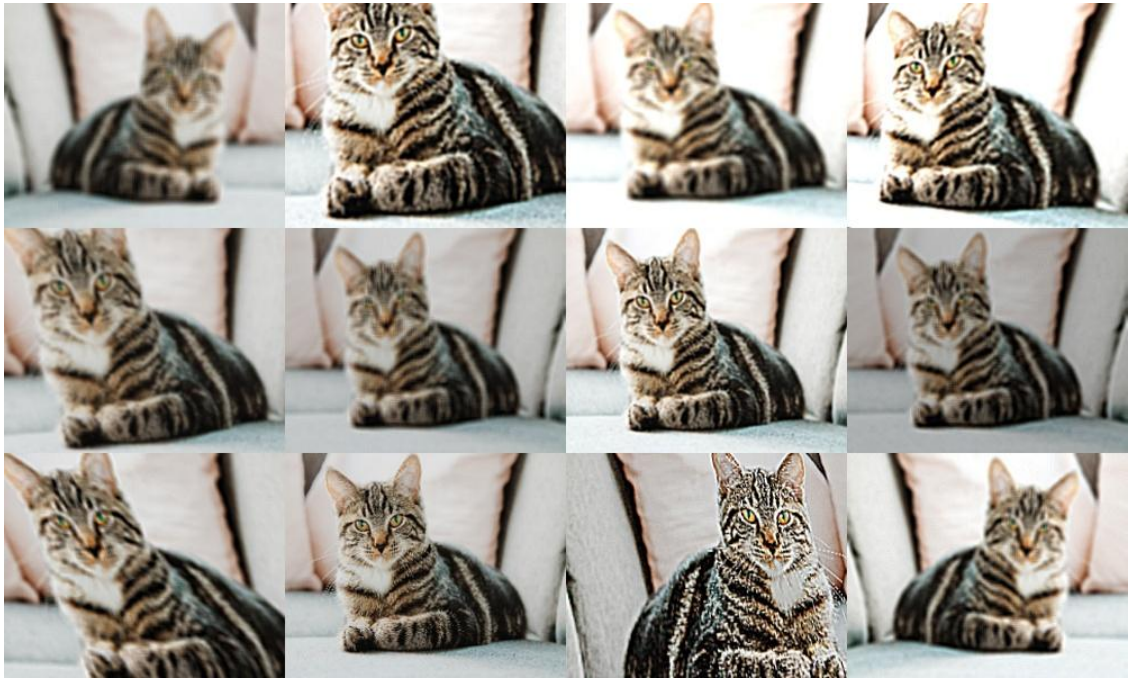
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ioffe et al. ICML 2015

Dataset Augmentation



Dataset Augmentation

More data is almost always a good way to improve a machine learning system to generalize better.

We can sometimes create new data by augmenting what data we have.

Image classification: We can apply many transformations to the images that do not change the image label. Rotate the image slightly, shift the mean intensity, scale the image, …

Image segmentation: Rotate and scale both the image and the ground truth segmentation in exactly the same way. Shift the mean intensity. Add noise to the image.

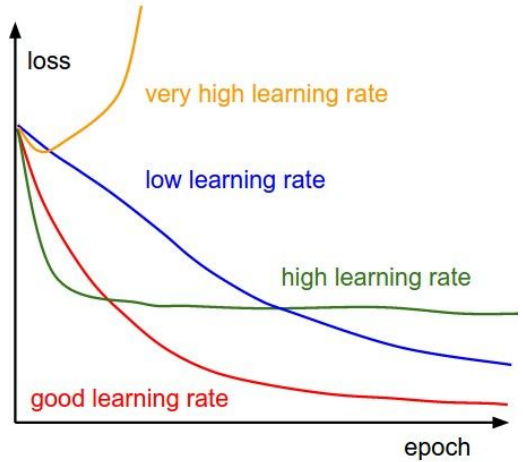
In general there are typically many invariants that computer vision systems should be able to handle. One must be careful to not apply transformations that makes the labelling incorrect. What about mirroring?

Hyperparameter Tuning

Learning rate: If the loss is diverging, decrease the learning rate. If the loss decreases slowly, you should probably increase the learning rate. If the loss has plateaued, decrease the learning rate.

Regularization: Is there a big gap between training and validation accuracy? Increase regularization.

Is the validation accuracy increasing? Keep training.



http://cs231n.github.io/neural-networks-3/

Weight Initialization

Xavier initialization - Sample from a Gaussian with mean 0 and variance $1/n_{in}$. Assuming independent and identically distributed variables, this keeps the variance constant through conv layers.

He initialization - Better coupled with relu-nonlinearities. Sample from a Gaussian with mean 0 and variance $2/n_{in}$. Unrealistic independence assumptions, but it works very well in practice.

We need to know n_{in}, the number of ingoing connection to an activation. What about a k x k conv filter where the previous layer has c channels? $n_{in}=k^2c$. Useful in assignment 3. Keep track of all sizes!

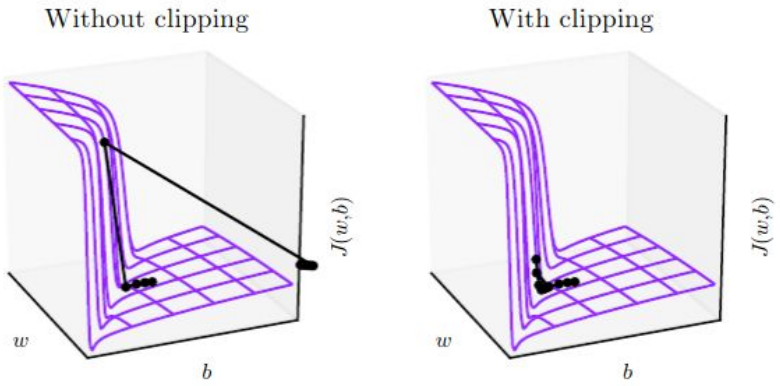
Bias elements are almost always initialized to 0.

Weight Clipping

Widely used for training recurrent neural networks.

The gradient specifies locally the best direction but not necessarily a good step size, especially not for non-convex functions.

Either constrain the norm of the gradient to be below a certain threshold or all elements in the gradient to be below a certain threshold.



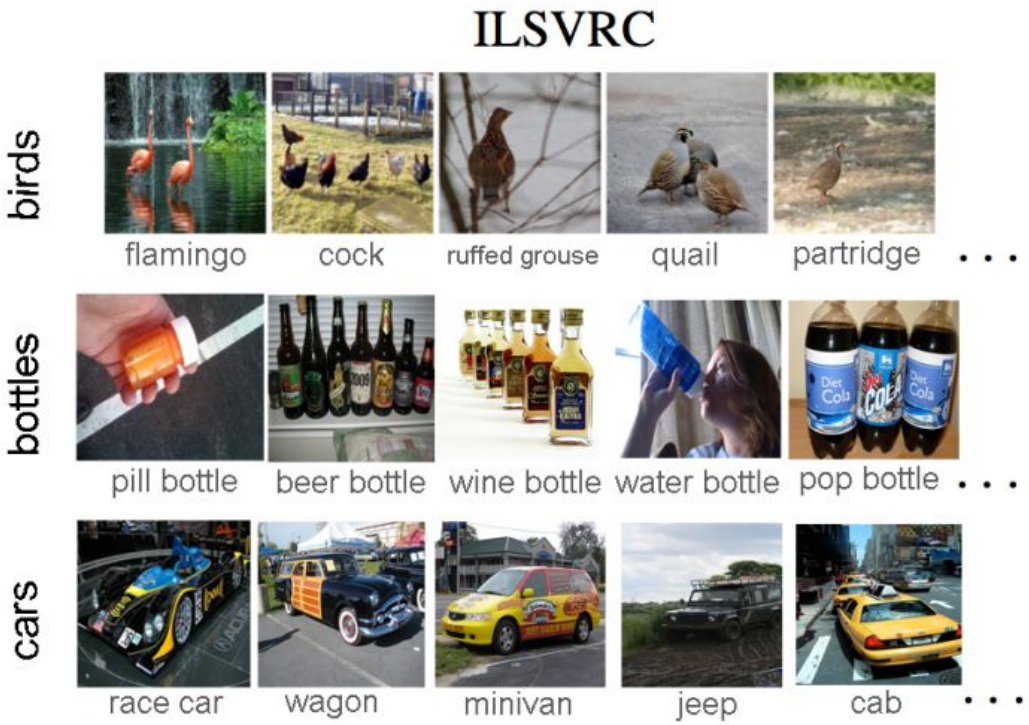
Pascanu et al. 2013

Imagenet

1,000 classes

~1,200,000 images, one label per image.

Fine grained classes as shown.

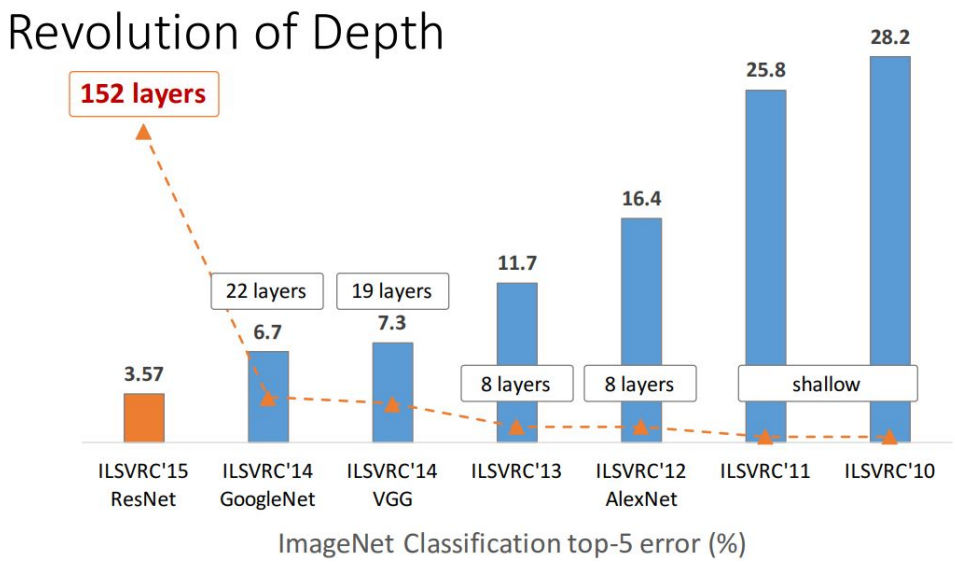


http://ai.stanford.edu/~olga/slides/ImageNetAnalysis_bavm_10_5_13.pdf

Imagenet

Since 2012, deep learning based methods have dominated image classification.

Percentage of test images where the correct label is among the top 5 predictions of the CNN.

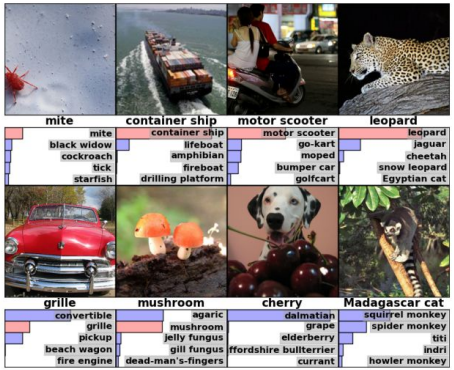


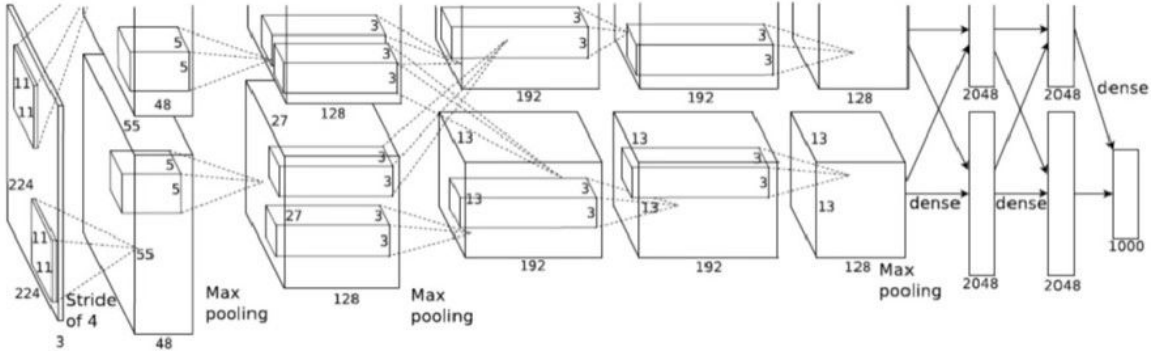
Alexnet

Start of the deep learning revolution

38,849 citations at the time of writing

(Conv + max pooling)·2 + 3 conv layers + max pooling + 3 fully connected layers + softmax classifier





Krizhevsky et al. NIPS 2012

VGG

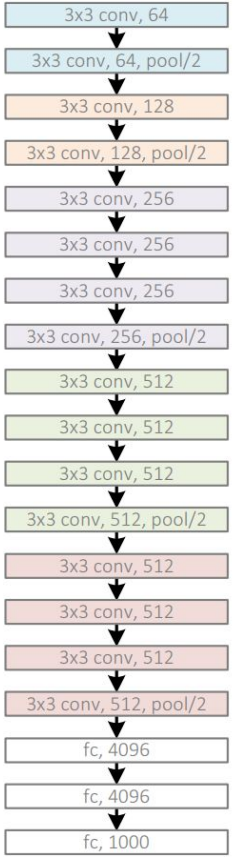
Only use 3x3 filters.

Use blocks of 3x3 filters, followed by a max pooling layer.

For each block, the spatial resolution is halved (i.e. 1 / 4 the number of pixels) and the number of channels doubles.

At the end there are three fully connected layers.

Very common to use the computed values of the last conv layer as feature extractor for models designed for other computer vision tasks.



http://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

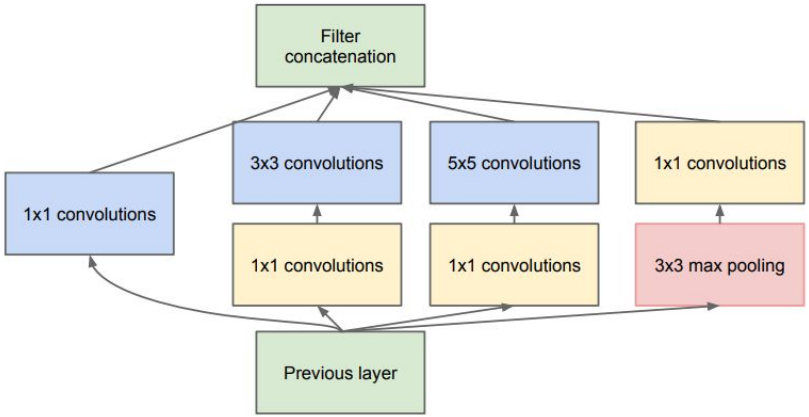
GoogleNet

Inception module.

We want to add more layers. What should we add? 1x1 convolutions? 3x3 convolutions? 5x5 convolutions? Max-pooling?

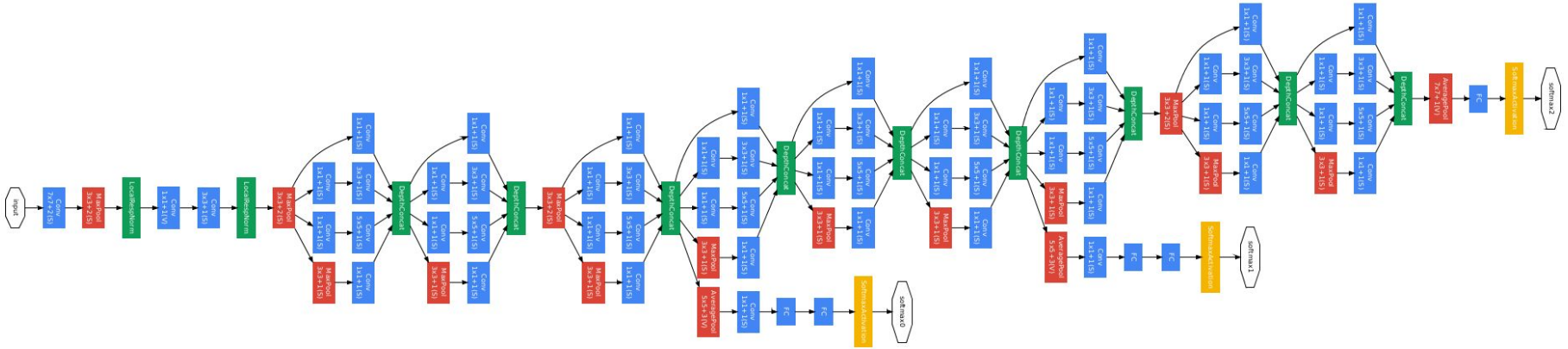
Add all of them!

Note that 1x1 convolutions act as dimensionality reduction filters if there are fewer output channels than input channels.



Szegedy et al. CVPR 2015

GoogleNet



Szegedy et al. CVPR 2015

ResNet

Clearly more layers are better from a representation capacity viewpoint. Why not just keep adding more?

Optimizing a neural network is hard! Note the ordering in the figure to the right.

Even though the network capacity is a lot higher with 56 layers the training error does not decrease with more layers.

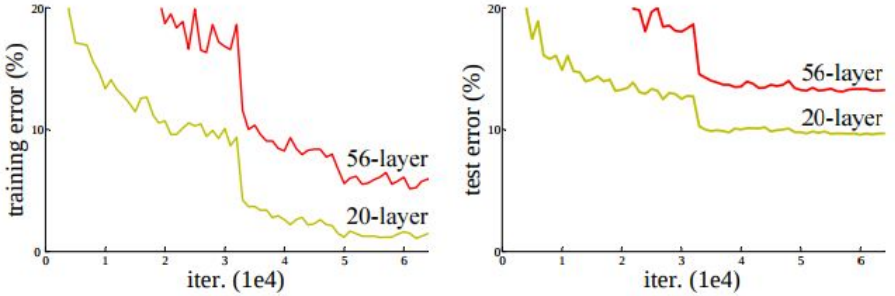


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

He et al. CVPR 2016

ResNet

Add residual correction at most layers in the VGG network.

Closer connection between early layers and where the loss is helps gradient flow and the network learns better.

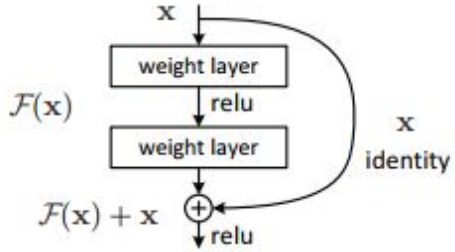


Figure 2. Residual learning: a building block.

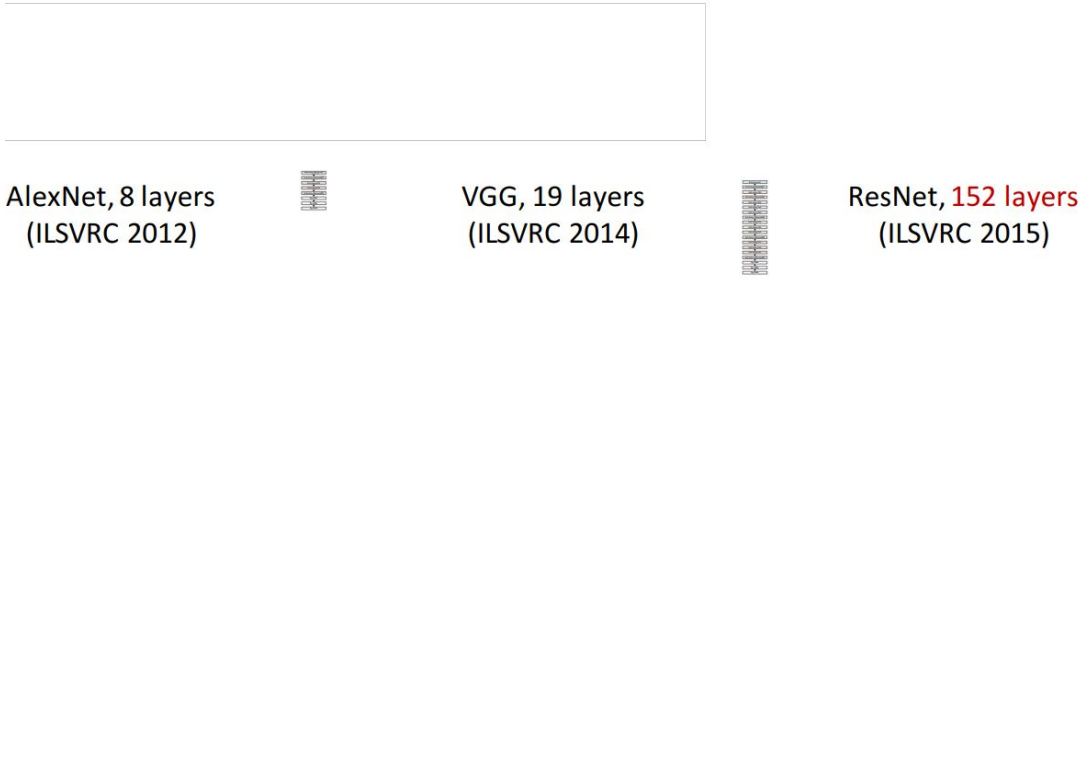
He et al. CVPR 2016

ResNet

He et al. CVPR 2016

Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

ResNet

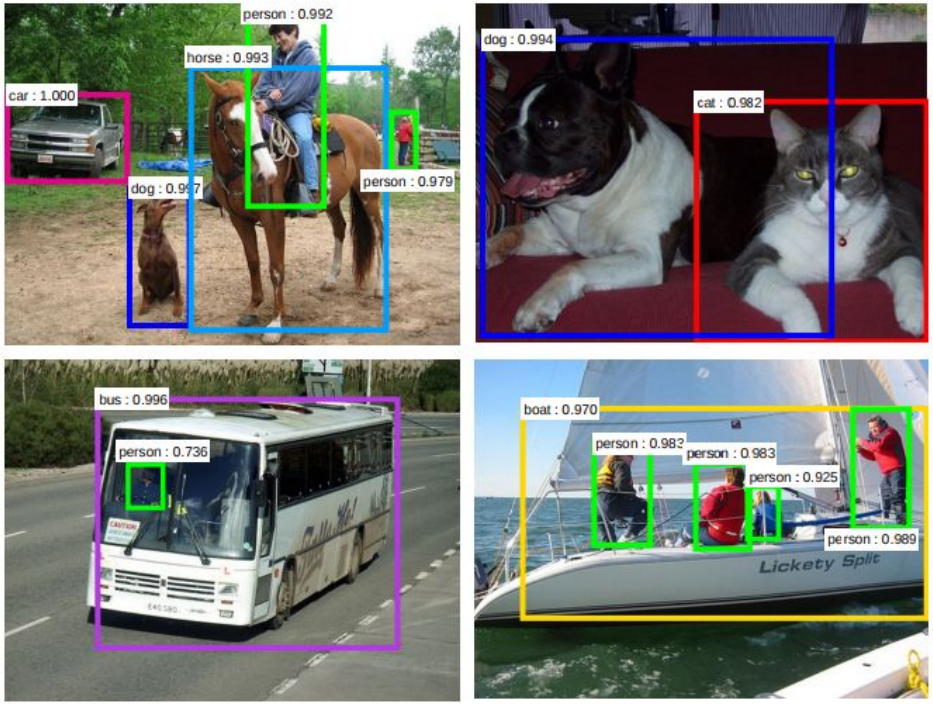

AlexNet, 8 layers
(ILSVRC 2012)
VGG, 19 layers
(ILSVRC 2014)
ResNet, 152 layers
(ILSVRC 2015)

Object Detection

One of the most studied problems in computer vision.

Find all objects in the image and mark them with a bounding box.

Often treated as two subproblems: Localization and classification.



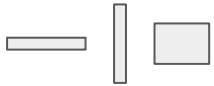
Ren et al. NIPS 2015

Faster R-CNN

Two stages: Region proposal networks and classifier branches.

Use a base network (VGG or ResNet) to extract features. For each position, predict, a) whether it is an object, b) displacement of the object relative reference bounding boxes.

Predict displacements
relative a few anchor boxes.



Pool interesting regions using NMS (non-max suppression)

Feed the regions of interest (RoIs) to a classifier, use NMS again to get the final predictions.

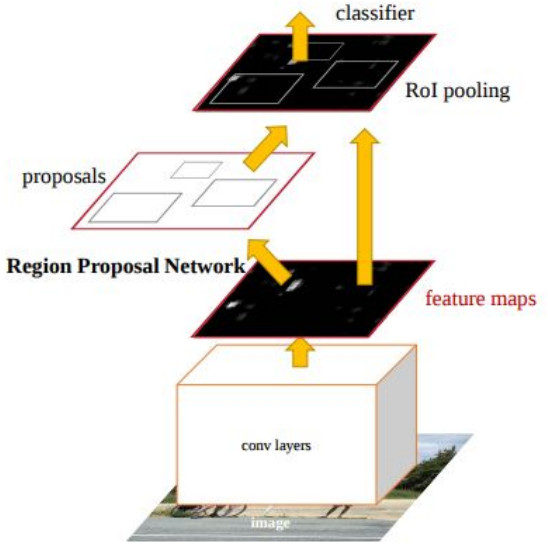


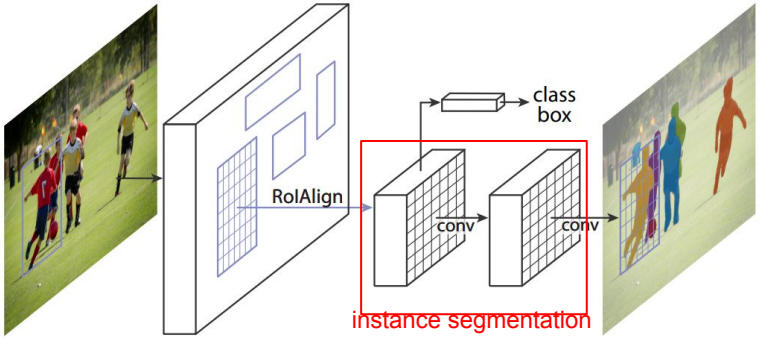
Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

Ren et al. NIPS 2015

Mask R-CNN

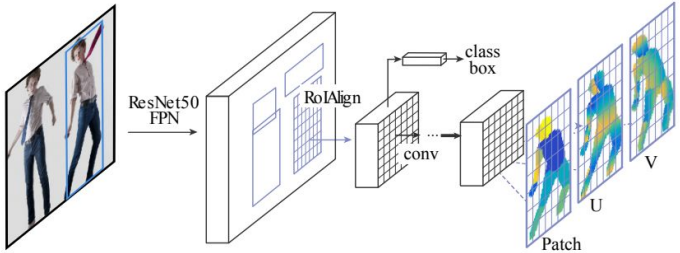
Instance level semantic segmentation

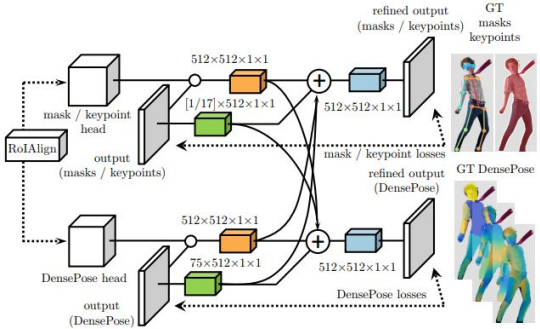
As faster R-CNN but it also predicts the object mask in the classification branch. The extension is marked in red.



He et al. ICCV 2017

Densepose



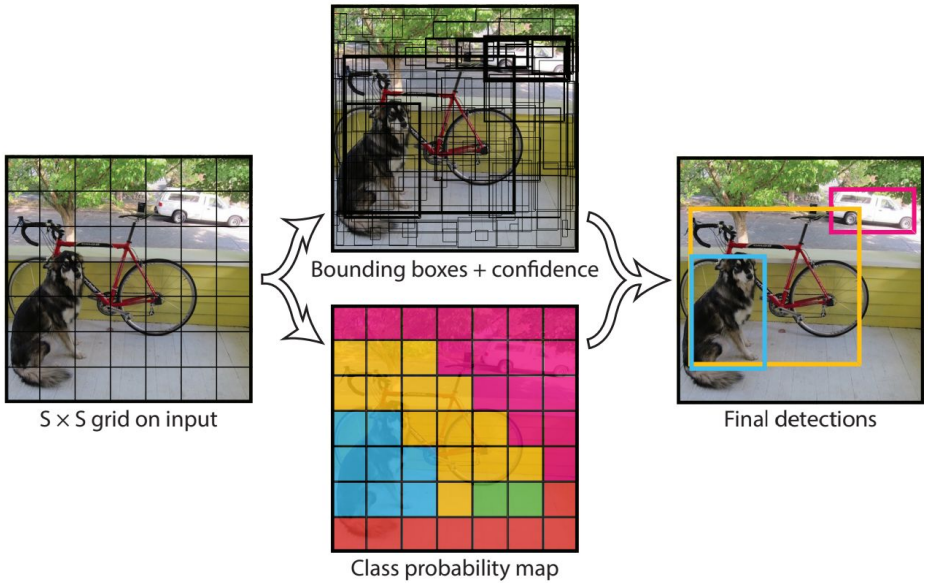


Güler et al. CVPR 2018

Yolo - You Only Look Once

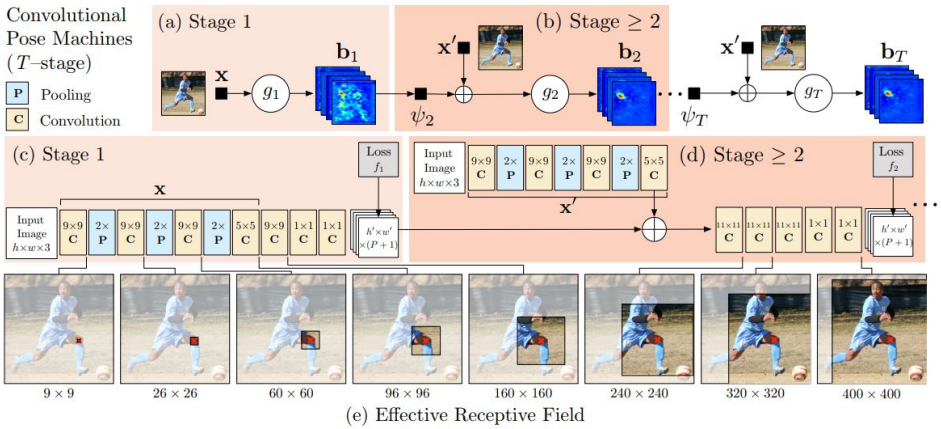
Predict the classes directly instead of pooling interesting regions and send to a classification network.

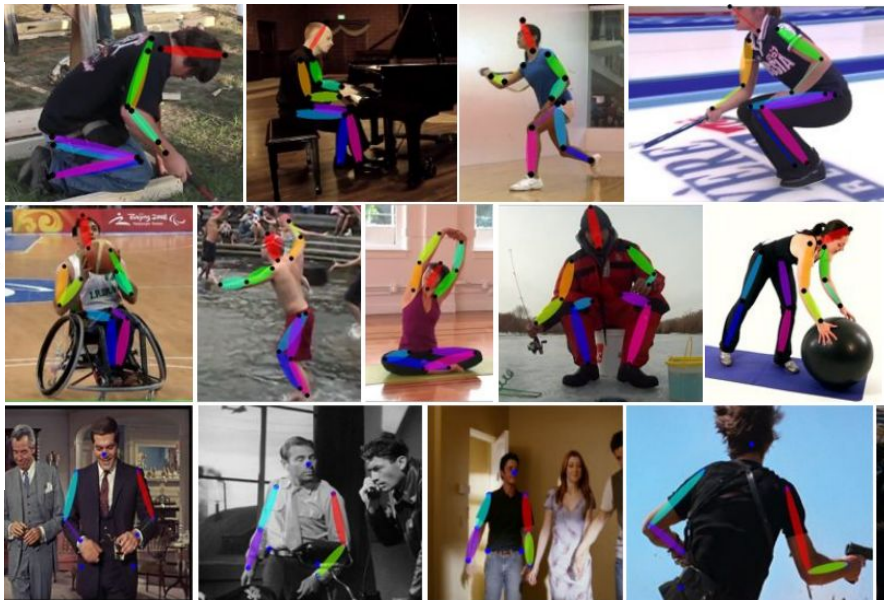
For each (sub-sampled) pixel or grid element, predict the class, and the displacement, the scalings and a confidence for each reference anchor box.



Redmond et al. CVPR 2016

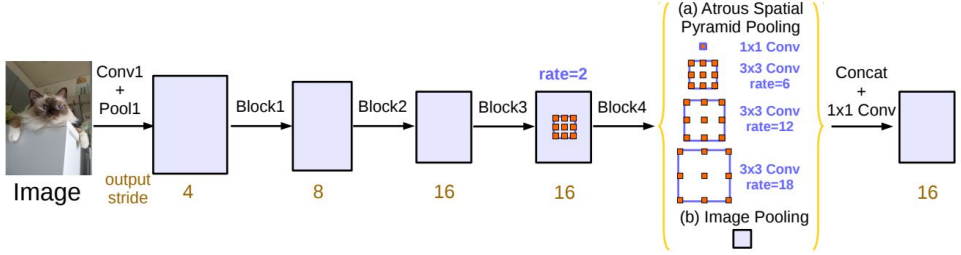
Human Pose Estimation





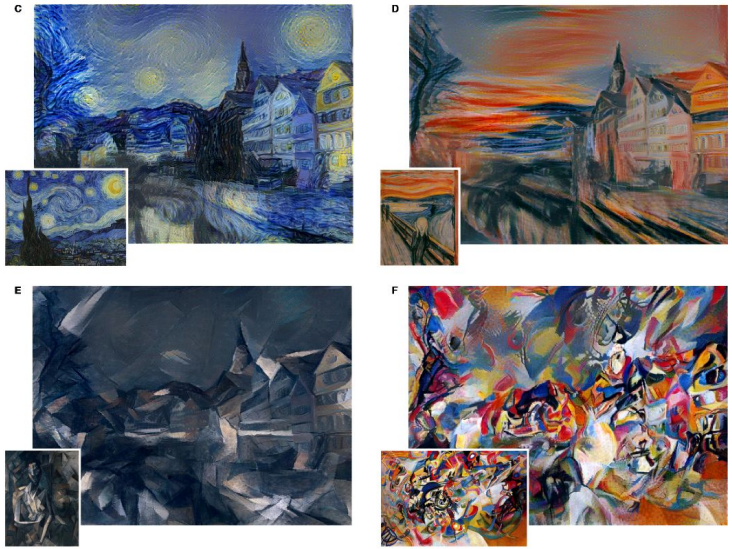
Wei et al. CVPR 2016

Semantic Segmentation - Deeplab-v3



https://arxiv.org/pdf/1706.05587.pdf

Neural Style Transfer



Gatys et al. CVPR 2016

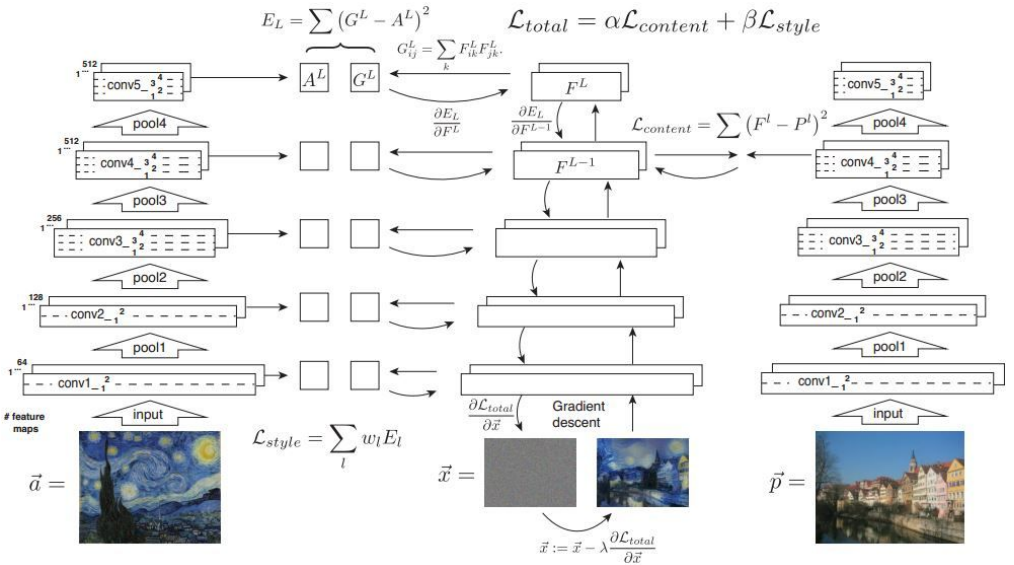
Neural Style Transfer

Intuition: Higher level features should remain the same as the original image while the lower level features should match the style image.

Optimize over the new, style transferred image. All other layers are frozen.

Completely dependent on the representational capability of the CNN it uses.

Backpropagate to the input image. The network weights are frozen while the image is optimized.



Gatys et al. CVPR 2016

Readings

Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning, MIT Press 2016 (online at http://www.deeplearningbook.org/)

Optimization

- Sections 8.1-8.5 and 8.7 in the deep learning book