# Convolutional Neural Networks and Supervised Learning

Eilif Solberg

August 30, 2018

# Outline

# Convolutional Architectures

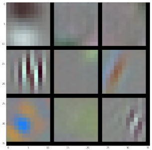
# Template matching


Figure: Illustration from
`http://pixuate.com/technology/template-matching/`

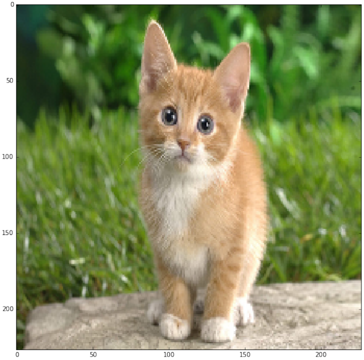
1. Try to match template at each location by "sliding over window"
2. Threshold for detection

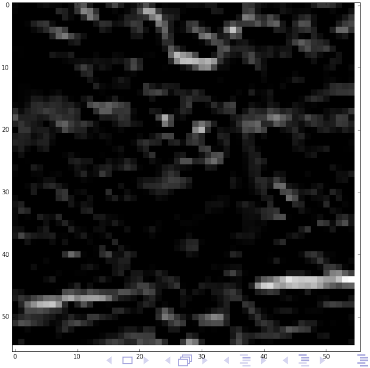
For 2D-objects, kind of possible but difficult

# Convolution



Which filter has produces the activation map on the right?



# Convolutional layer

–> Glorified template matching

- Many templates (aka output filters)
- We *learn* the templates, the *weights* are the templates
- Intermediate detection results only *means to an end*
  - treat them as *features*, which we again match new templates to
- Starting from the second layer we have “nonlinear filters”

# Hyperparameters of convolutional layer

1. Kernel height and width - template sizes
2. Stride - skips between template matches
3. Dilation rate
   - "Wholes" in template where we don't care
   - Larger field-of-view without more weights...
4. Number of output filters - number of templates
5. Padding - expand image, typically with zeros

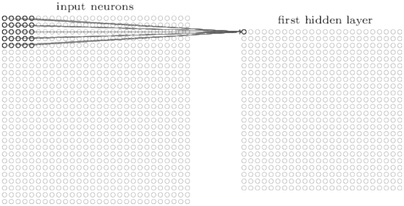



Figure: Image from http://neuralnetworksanddeeplearning.com/

# Detector / activation function

- Non-saturating activation functions as ReLU, leaky ReLU dominating

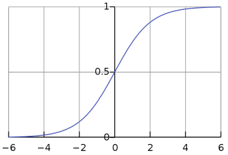



Figure: Sigmoid function

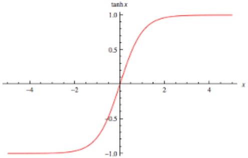


Figure: Tanh function

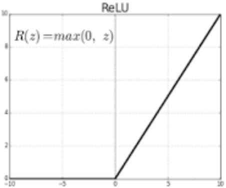



Figure: ReLU function

# Basic CNN architecture for image classification

Image –> [Conv –> ReLU]xN –> Fully Connected –> Softmax

- Increase filter depth when using stride

Improve with:

- Batch normalization
- Skip connections ala ResNet or DenseNet
- No fully connected, average pool predictions instead

# Training

# How do we fit model?

How do we find parameters $\theta$ for our network?

# Supervised learning

- Training data comes as $(X, Y)$ pairs, where $Y$ is the target
- Want to learn $f(x) \sim p(y|x)$, conditional distribution of $Y$ given $X$
- Define *differentiable* surrogate loss function, e.g. for a single sample

$$l(f(X), Y) = (f(X) - Y)^2 \text{regression} \tag{1}$$

$$l(f(X), Y) = -\sum_c Y_c log(f(X)_c) \text{classification} \tag{2}$$

# Gradient

- The direction for which the function increases the most

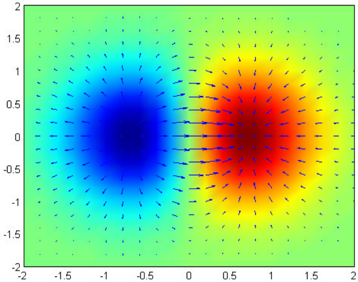



Figure: Gradient of the function $f(x^2, y^2) = x/e^{x^2+y^2}$ [By Vivekj78 [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], from Wikimedia Commons]

# Backpropagation

- Efficient bookkeeping scheme when applying chain rule for differentiation
- Biologically implausible?

# (Stochastic) gradient descent

Taking steps in the opposite direction of the gradient

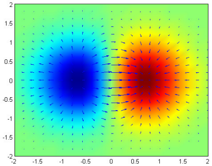


Figure: [By Vivekj78 [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], from Wikimedia Commons]

- Full gradient too expensive / not necessary

$$\sum_{i=1}^{N} \nabla_\theta l(f(X_i), Y_i) \approx \sum_{i=1}^{n} \nabla_\theta l(f(X_{P(i)}), Y_{P(i)}) \tag{3}$$

for a random permutation $P$.

Many different extensions to standard SGD

- SGD with momentum, RMSprop, ADAM

# Network, loss, optimization

- Weight penalty added to loss term, usually squared L2 normalization uniformly for all parameters

$$l(\theta) + \lambda \|\theta\|_2^2 \tag{4}$$

- Dropout
- Batch normalization
  - Intersection of optimization and generalization
  - Your best friend and your worst enemy

# More on batch normalization

For a tensor [batch_size $\times$ height $\times$ width $\times$ depth], normalize "template matching scores" for each template $d$ by

$$\mu_d \leftarrow \frac{1}{N * H * W} \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{i,h,w,d} \tag{5}$$

$$\sigma_d^2 \leftarrow \frac{1}{N * H * W} \sum_{i=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{i,h,w,d} - \mu_d)^2 \tag{6}$$

$$\hat{x}_{i,h,w,d} \leftarrow \frac{x_{i,h,w,d} - \mu_d}{\sqrt{(\sigma_d^2 + \epsilon)}} \tag{7}$$

$$y_{i,h,w,d} \leftarrow \gamma \hat{x}_{i,h,w,d} + \beta \tag{8}$$

where $N$, $H$ and $W$ represents batch size, height and width.

- "Template/Feature more present *than usual* or not"
- During inference we use stored values for $\mu_d$ and $\sigma_d$.

# Data augmentation

Idea: apply random transformation to $X$ that does not alter $Y$.

- Normally you would like result $X'$ to be *plausible*, i.e. *could* have been a sample from the distribution of interest
- Which transformation you may use is application-dependent.

## Image data

- Horizontal mirroring (issue for objects not left/right symmetric)
- Random crop
- Scale
- Aspect ratio
- Lightning etc.

## Text data

- Synonym insertion
- *Back-translation*: translate and translate back with e.g. Google Translate!!!

# Hyperparameters to search

- Learning rate (and learning rate schedule)
- Regularization params: L2, (dropout)

# Search strategies

- random search rather than grid search
- logscale when appropriate
- careful with best values on border
- may refine search

# Achitecture search

# Architecture search

1. Define the search space.
2. Decide upon the optimization algorithm
   - random search, reinforcment learning, genetic algorithms

# Neural architecture search

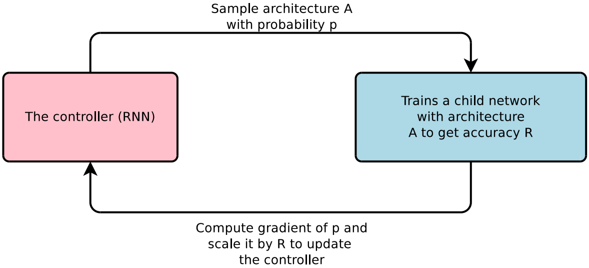



Figure: An overview of Neural Architecture Search. Figure and caption from [?].

# NAS1 - search space

Fixed structure:

- Architecture is a series of layers of the form

$$\text{conv2D(FH, FW, N)} \longrightarrow \text{batch-norm} \longrightarrow \text{ReLU}$$

Degrees of freedom:

- Parameters of conv layer
  - filter height, filter width and number of output filters
- Input layers to each conv layer

# NAS1 - discovered architecture

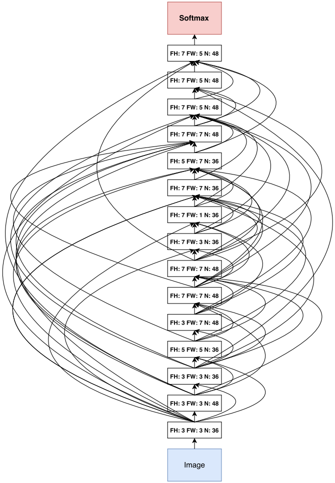



Figure: FH is filter height, FW is filter width and N is number of filters. If one layer has many input layers then all input layers are concatenated in the depth dimension. Figure from [?].

# NAS2 - search space

Fixed structure:

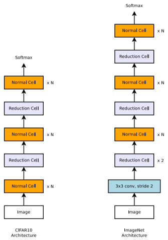



Figure: Architecure for CIFAR-10 and ImageNet. Figure from [?].

Degrees of freedom:

- Some freedom in *normal cell* and *reduction cell*, shall see soon

# NAS2 - discovered convolutional cells

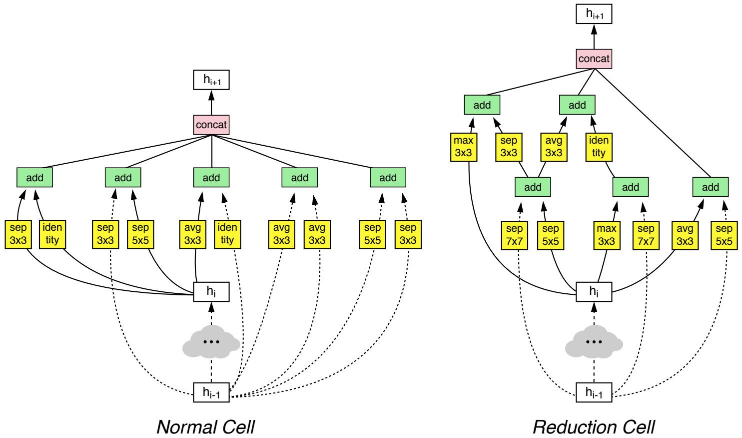



Figure: NASNet-A identified with CIFAR-10. Figure and caption from [?].

# NAS2 - Performance(computational_cost)

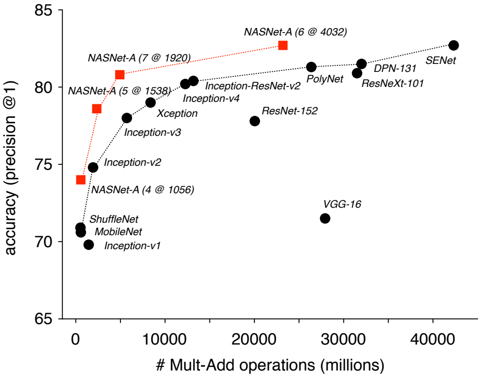



Figure: Performance on ILSVRC12 as a function of number of floating-point multiply-add operations needed to process an image. Figure from [?].

# NAS2 - Performance(#parameters)

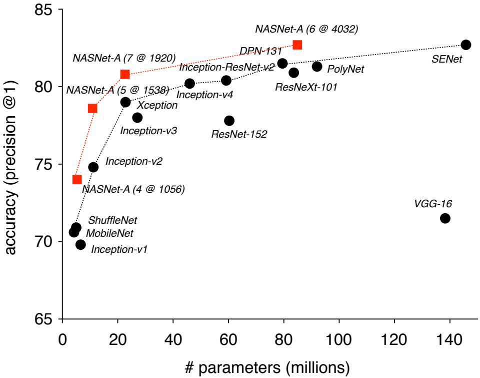



Figure: Performance on ILSVRC12 as a function of number of parameters. Figure from [?].

# Bibliography

# Bibliography I