Computer Vision
Learning

dsai.asia

Asia Data Science and Artificial Intelligence Master's Program

DS&AI

Co-funded by the
Erasmus+ Programme
of the European Union

Readings

Readings for these lecture notes:

- Viola, P., and Jones, M.J. (2001), Rapid object detection using a boosted cascade of simple features. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pages 511–518.
- Dalal, N., and Triggs, B. (2005), Histograms of oriented gradients for human detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, 886–893.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016), *Deep Learning*. MIT Press, Chapter 9.
- Krizhevsky, A., Sutskever, I., and Hinton, G.E. (2012), ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105.

Readings

- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015), Going deeper with convolutions. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016), Rethinking the Inception architecture for computer vision. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016), Deep Residual Learning for Image Recognition, In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.
- Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2017), Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *Association for the Advancement of Artificial Intelligence Conference on AI (AAAI)*, 4278–4284.

Readings

Readings, continued...

- Hu, J., Shen, L., and Sun, G. (2018), Squeeze-and-excitation networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7132–7141.
- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016), You only look once: Unified, real-time object detection
- He, K., Gkioxari, G, DollÃ¡r, P., and Girshick, R. (2017), Mask R-CNN
- Redmon, J. and Farhadi, A. (2018), YOLOv3: An incremental improvement
- Chen, X., Girshick, R., He, K., and DollÃ¡r, P. (2019), TensorMask: A Foundation for Dense Object Segmentation

Outline

Next we focus on algorithms for learning in vision systems.

Classically, machine learning comprises three basic problems:

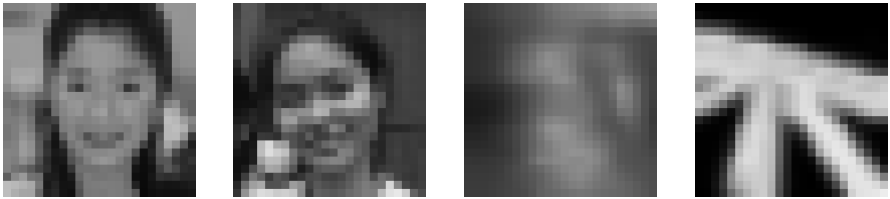
- Classification: place instances into one or more of a set of given discrete categories.
- Regression: estimate a function from sample inputs/outputs that can later be used for interpolation or extrapolation.
- Density estimation: estimate a probability density function from a sample from the distribution that can later be used, e.g., for anomaly detection.

Try to think of an example of each type of learning that would be useful in a vision system.

So far we've seen a lot of techniques for determining where something is.

Although all three classic learning problems are useful, we will focus mainly on classification, i.e., knowing what something is.

Example: determining if a 24×24 image patch is a face or a non-face.



As an example of how we might approach object finding and recognition, consider template matching:

- We are looking for some kind of object in an image.
- But we don't know where the object is.
- We might sweep a detection window over the image and classify each possible window.
- But we don't know what orientation the object is in.
- For each detection window, we might consider several possible orientations.
- But we don't know how big the object is.
- We might search not only the original image, but successively downscaled versions of the image.

This is the template matching approach to object recognition.

Template matching can get the job done but is plagued with problems:

- Speed
- Which examples to use as templates
- How to evaluate similarity between template and image region?

In the learning approach, we let the data decide.

Necessary ingredients for classification

In most cases, finding and recognizing objects boils down to classification. Classification involves two steps:

- Inference: determining a probability distribution over set of hidden variables given a set of observed variables.
- Decision making: determining what to do based on the inferred probability distribution.

Probabilistic or generative methods make these two steps explicit. Example: maximum a posteriori classification of pixels as skin or not skin.

Black box methods combine the two steps in an opaque fashion. Example: support vector machine or neural network classification of faces.

Both types of classifier require some form of learning.

Learning

Learning

Estimating the parameters and/or structure of a probabilistic model from a training set.

The learning approach is immensely powerful. Consider the case of speech recognition (from Bill Freeman):

- Prior to 1980s, speech recognition techniques were ad hoc.
- The introduction of statistical models with efficient learning algorithms (HMMs) revolutionized the field.
- Now the person with the best training set wins.

The same transformation began in computer vision starting around 2000 and is complete today.

The 2010's have seen big changes machine learning applications in vision.

Prior to 2012 or so: feature based approaches were the winners for most problems.

Feature-based methods work in two phases:

- Given an instance, compute a low-dimensional feature vector describing the instance.
- Pass the feature vector to a classifier.

The classifier might be a SVM, a Gaussian classifier, a shallow neural network, or a nearest neighbor classifier.

Since 2012, deep learning models have completely transformed the field.

The general idea is to move as close as possible to end-to-end learning:

- The input is handed directly to the model without preprocessing.
- The model outputs the decision, possibly after a complex, multi-step calculation.

Successful applications that seemed insanely futuristic just a few years ago include self-driving cars and payments using face recognition.

As of the late 2010's, most deep learning models require parallel computation with GPUs for real time performance.

This gives us a simple decision tree:

- If a simple feature-based model is sufficient for a given task, go with it.
- Otherwise, if you have sufficient training data and sufficient compute power at runtime, deep learning will win every time.
- Otherwise, if you have insufficient training data but still have sufficient compute power at runtime, you may be able to fine-tune an existing deep learning model.
- Otherwise, hand-craft the best feature-based model you can.

The most important applications of supervised machine learning in computer vision are probably detection, classification, and segmentation.

Detection is really just classification:

- We may classify each subwindow of an image sequentially using a sweep window.
- We may look for "interesting" locations first using some kind of interest operator.
- Or, we may classify all possible locations in parallel.

Similarly, segmentation is just classification:

- We may classify each pixel as a member of the same segment as its neighbors or a different segment.
- In semantic segmentation, we attach a specific category label to each pixel of the image.

The key in all of these cases is to be able to classify an image or a patch of an image or a pixel of an image.

We'll thus first look at some different kinds of simple classifiers then see how they are applied to machine vision problems.

Outline

There are two distinct kinds of classifier induction algorithms:

- Explicit probability models or generative models use the training set to build a probability model for the likelihood $P(\mathrm{x} \mid y)$ or the posterior $P(y \mid \mathrm{x})$.

  Example: estimate the parameters of a Gaussian distribution from each example set.

  Limitation: if the probability model is not sufficiently powerful to capture the true distribution of the members of each class, we might not get a good decision boundary.

- Direct decision boundary induction tries to find a good decision boundary without the intermediate step of probability modeling.

  Example: support vector machines (SVMs).

  Limitation: if the induction algorithm is not powerful enough to model the decision boundary accurately, we might not get a good decision boundary. If it is too powerful it might pay too much attention to the training set without generalizing.

As an explicit probabilistic model, the multidimensional Gaussian density is

$$P(\mathrm{x}) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathrm{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathrm{x}-\boldsymbol{\mu})\right)$$

where $\boldsymbol{\mu}$ is the mean and Σ is the covariance of the distribution. For a N-class classification problem, we have distinct means and covariances $\boldsymbol{\mu}_k, \Sigma_k, k \in 1 \ldots N$. Using the training set we estimate the parameters of the Gaussian class-conditional densities $P(\mathrm{x} \mid k)$:

$$\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_{i=1}^{N_k} \mathrm{x}_{ki} \qquad \Sigma_k = \frac{1}{N_k - 1}\sum_{i=1}^{N_k}(\mathrm{x}_{ki}-\boldsymbol{\mu}_k)(\mathrm{x}_{ki}-\boldsymbol{\mu}_k)^T$$

Now what is the appropriate decision rule? Hint: if $a < b$, $\log a < \log b$.

As an example of a direct decision boundary induction technique, consider nearest neighbors:

(k, l) nearest neighbors algorithm

Find the k nearest neighbors of x in the training set, and return the most common category in the set, if it appears at least l times.

Nearest neighbors often performs very well, without having to model the data's probability distribution.

Nearest neighbors has one main limitation: computing Euclidean distance to every element of the training set is expensive, especially if the feature space is high dimensional.

Research issues include avoiding calculating distances to every element of the training set, what distance measures are appropriate, and how to remove superfluous training set items.

One example was Beis and Lowe's "best bin first" approximate nearest neighbors algorithm used to quickly get correspondences for SIFT keypoints.

As another example of an explicit probabilistic method, consider a Bayesian skin pixel classifier:

- Training: Construct a histogram of skin pixels and non-skin pixels in HS space.
- Classification: Classify each pixel as skin or non-skin based on which model has the highest histogram count.

The skin histogram gives an estimate of $P(\mathrm{x} = (h, s)^T \mid \text{skin})$ and the non-skin histogram gives an estimate of $P(\mathrm{x} = (h, s)^T \mid \text{not skin})$.

Picking the category with the highest histogram count corresponds to Bayesian classification under uniform category priors.

Outline

How well are wee doing in a classification task?

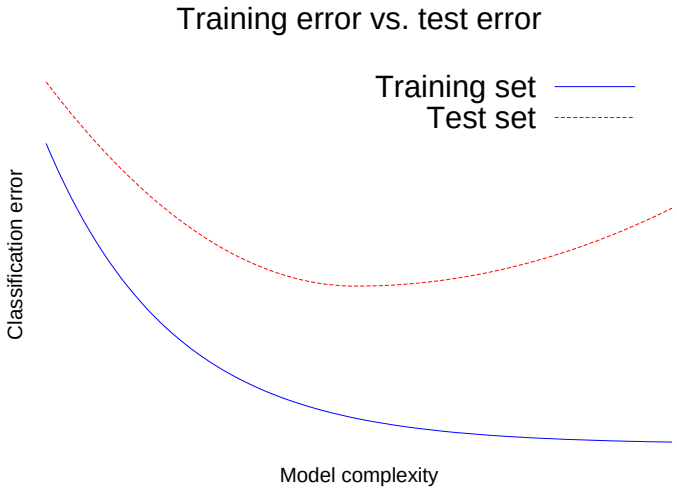
Training set loss/accuracy is one measure, but if the model has high VC dimension, it will tend toward overfitting.

There are many techniques for avoiding overfitting (early stopping, weight decay, drop-out, etc.).

However, to determine if these techniques are working properly, we need to measure the degree to which we are overfitting.

The main technique for measuring overfitting is to reserve a test set that is not used during training.

The test set should be independent of the training set. It will indicate how well our system will generalize to new unseen data.


Training error vs. test error
Training set
Test set
Classification error
Model complexity

Most models have two kinds of parameters: learned or tuned parameters and hyperparameters or free parameters.

Learned parameters include weights in a neural network, the elements of the hyperplane's normal vector in a SVM, and each frequency in a histogram.

Hyperparameters include the number of units/layers in a neural network, the error tolerance of a SVM, and the number of bins in the histogram.

To measure and control overfitting of the tuned parameters of a model, we need a test set.

To compare different models with different hyperparameters, we need a test set.

In practice, then, we need three data sets:

- Training set: the data set on which we minimize the cost function, by gradient descent or another method.
- Validation set: an independent data set used for estimating the generalization accuracy of a trained model, measuring overfitting, and performing early stopping.
- Test set: an independent data set used to compare different models with different hyperparameters.

Note, however, that if you use your test set more than once, it becomes a kind of training/validation set, and you are performing data snooping.

To be completely fair, then, it might be wise to reserve a fourth, final final test set representing unseen data that is only used once, when model selection is complete.

Reserving a test set wastes valuable training items, so cross validation is useful when data is scarce.

The best way to improve performance is almost always to increase the size of the training set.

But collecting new training data can be expensive, and training on unneeded data might waste time.

Bootstrapping selects new items for training according to how close they are to the existing classifier's decision boundary.

Outline

In a histogram-based skin classification problem, we might decide to build our histograms in the H-S plane of the HSV color model.

For more complex problems, how do we decide what features to use for our classifier?

1. Hand-crafted features: SIFT descriptors, principal components, local receptive field histograms, edge filter histograms, 30+ years of research and thousands of choices!
2. End-to-end feature learning with deep learning: Give the data complete control over what the features should be.
3. (A middle ground) Feature selection from a family of features: Put some intelligent design into a large family of possible features, then let the data decide which ones to use.

Feature selection is appropriate when runtime resources are constrained or the labeled data you have access to is insufficient for end-to-end learning.

Feature selection prinicples

- Find features related to the classification you want to make.
- Keep the number of features as small as possible.
- Do not use correlated features.

The simplest features are linear combinations of specific image pixels. Linear combinations are good computationally.

- Or we could do linear dimensionality reduction to transform a large number of correlated inputs into a smaller feature space.
- We could learn linear features that are good for a classification task.

Next we will see how to learn features linear features in the context of a face detection task.

This technique of learning fast linear features is due to Viola and Jones.

The combination of fast linear features, ensemble learning, and a decision cascade enabled the first real-time face detector, on 2000-era hardware.

The principles are still valid today.

We already discussed how object detection can be considered a classification problem.

We slide a detection window at multiple scales over the image, then for each window, we extract a set of simple features and classify the result as a positive or negative.

Feature selection and feature learning

Viola and Jones

Viola and Jones' famous method for face detection was the first real time face detection method.

Viola, P., and Jones, M.J. (2001), "Robust real-time face detection," *International Journal of Computer Vision* 57(2):137–154.

The method is based on learning of features for a sliding window classifier.

It builds a strong classifier by combining many simple classifiers called decision tree stumps, which use just a single feature and are extremely easy to train.

By building a powerful classifier out of an ensemble of simple classifiers using a single feature each, the method performs feature selection at the same time as it is learning the decision boundary.

Before we can understand their method, we need to understand the underlying learning algorithm, AdaBoost.

Simple direct decision boundary induction approaches like the linear SVM do not work for complex problems with many features and nonlinear decision boundaries.

To address this issue we have two directions to go in:

- Build a more complex model using a more powerful classifier such as a neural network or a kernelized support vector machine;
- Stick with a simple classifier, but use more than one classifier to make up for the deficiencies of a single, simple classifier.

Benefits of multiple simple classifiers:

- The individual classifiers are easy and quick to train.
- Simplicity means there is less risk of overfitting the training set than with a universal classifier.
- Feature selection is built in (for some classifiers).

There are two basic approaches to combining classifiers. First we train L different models, then at classification time, we either

- run the input pattern x through all classifiers and somehow average their predictions, or
- use x to select just one of the L classifiers to perform the job.

We focus on a variant of the first idea, a committee.

The idea of a committee is to average the predictions of a set of individual models.

We could, for example, train multiple classifiers, each using a different data set, then put them together.

The goal is for the errors of different classifiers to average out.

The trouble with the committee idea is that we only have one data set, and we would like to train on all of the data.

How can we get different committee members that are really different from each other?

One way, called bootstrapping or bagging, is to randomly sample N items from our N-item data set X with replacement.

Repeating this M times, we get M different data sets, which we can then use to train M different classifiers $y_m(\mathrm{x})$. Then the final combined model is

$$y_{COM}(\mathrm{x}) = \frac{1}{M} \sum_{m=1}^{M} y_m(\mathrm{x})$$

It can be shown that in bagging, if the errors made by the different models are uncorrelated and the average expected error of the individual classifiers is E_{AV}, then the expected error of the combined model is $\frac{1}{M}E_{AV}$.

The gain is huge, but in practice the assumption of uncorrelated errors does not hold.

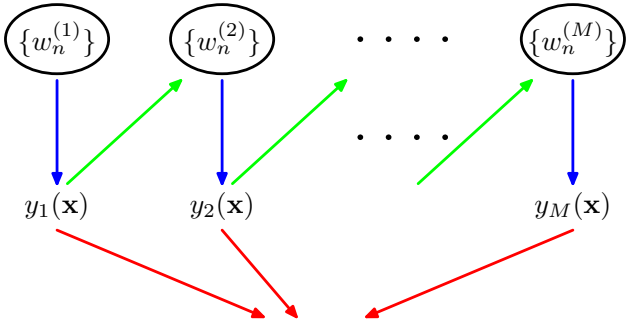
Boosting methods are a more sophisticated means to introducing variance among committee members.

Boosting combines multiple base classifiers to produce a committee whose performance is possibly better than any of the individual committee members.

The most widely used boosting method is Freund and Schapire's (1996) AdaBoost (adaptive boosting) algorithm.

Boosting combines multiple weak learners whose performance need only be slightly better than random.

The main difference between boosting and bagging is that the learners are trained sequentially, where each classifier is trained on a weighted form of the training set, and the weights depend on the performance of the previous classifiers.



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_m^M \alpha_m y_m(\mathbf{x})\right)$$

Bishop (2006), Fig. 14.1

The scheme:

- We have a two-class classification problem with training data $\mathrm{x}_1, \ldots, \mathrm{x}_N$ and corresponding binary targets $t_1, \ldots, t_N$ where $t_n \in \{-1, 1\}$.
- We additionally give each data point x_n a weighting parameter w_n which is initialized to $1/N$ for all points.
- Suppose we have a training procedure that produces a base classifier $y(\mathrm{x}) \in \{-1, 1\}$ using the weighted data.
- At each stage, AdaBoost trains a new classifier using weights that are adjusted to give more weight to points misclassified by the previous classifier.
- Finally the base classifiers are combined into a committee with weighted majority voting.

AdaBoost: Objective

Given a training data set $\mathrm{x}_1, \ldots, \mathrm{x}_N$, corresponding binary targets $t_1, \ldots, t_N$ where $t_n \in \{-1, 1\}$, and a method for constructing a weak classifier minimizing the weighted classification error, construct a strong classifier $Y_M(\mathrm{x})$.

AdaBoost: Algorithm (Bishop, 2006, p. 658)

(i) Initialize weights $\{w_n\}$ by setting $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$.

(ii) For $m = 1, \ldots, M$:

(a) Fit a classifier $y_m(\mathrm{x})$ to the training data by minimizing the weighted error function

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathrm{x}_n) \neq t_n).$$

Here $I(\cdot)$ is the indicator function, equal to 1 when its argument is true and equal to 0 otherwise.

AdaBoost: Algorithm (Bishop, 2006, p. 658, continued)

(ii) continued:

(b) Evaluate the quantities

$$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathrm{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

then use them to evalulate

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}.$$

(c) Update the weights

$$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(y_m(\mathrm{x}_n) \neq t_n)\}$$

AdaBoost: Algorithm (Bishop, 2006, p. 658, continued)

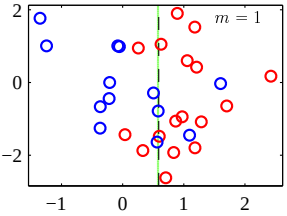
(iii) Make predictions using the final model

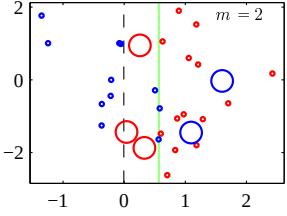
$$Y_M(\mathrm{x}) = \mathrm{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathrm{x})\right).$$

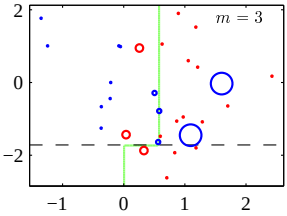
Some intuitive observations about the algorithm:

- The first classifier $y_1(\mathrm{x})$ is trained with uniform weights, so it will be a typical classifier trained on X.
- In step (ii)(c), we see that $w_n^{(m)}$ is held constant for cases where $y_m(\mathrm{x}_n) = t_n$ and increased for cases where $y_m(\mathrm{x}_n) \neq t_n$.
- This forces later classifiers to put more emphasis on training examples that have been misclassified by previous classifiers.
- ϵ_m represents the weighted error measure for classifier m.
- In step (ii)(c), ϵ_m is used to give even more weight to misclassified examples when $y_m(\mathrm{x})$ is good.
- In step (iii), ϵ_m is used to give a bigger vote to classifiers that had low error during training.

Here is an example with decision stump learners. Color represents class and radius represents weight. The dashed boundary is the decision surface for the mth classifier, and the green boundary is the decision surface for the combined classifier.



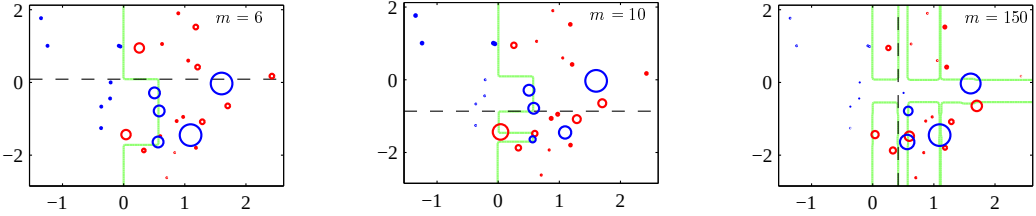




Bishop (2006), Fig. 14.2(a)-(c)

Feature selection and feature learning

Example



Bishop (2006), Fig. 14.2(d)-(f)

Friedman et al. (2000) found a straightforward explanation of AdaBoost.

Suppose we have an error function

$$E = \sum_{n=1}^{N} \exp\{-t_n f_m(\mathrm{x}_n)\}$$

where $f_m(\mathrm{x})$ is the linear combination of base classifiers

$$f_m(\mathrm{x}) = \frac{1}{2} \sum_{l=1}^{m} \alpha_l y_l(\mathrm{x})$$

and $t_n \in \{-1, 1\}$ are the targets. What if we minimize E with respect to weighting coefficients α_l and parameters of $y_l(\mathrm{x})$?

It turns out that if we fix the base classifiers $y_1(\mathrm{x}), \ldots, y_{m-1}(\mathrm{x})$, we find that $y_m(\mathrm{x})$, α_m, and $w_n^{(m+1)}$ as defined by AdaBoost exactly minimize E (see Bishop, 2006, pages 659–661).

This exponential error function is different from the cross-entropy error function typically used in machine learning:

$$E = -\sum_{n=1}^{N} \{t_n \ln y(\mathrm{x}_n) + (1 - t_n) \ln(1 - y(\mathrm{x}_n))\}$$

with targets $t_n \in \{0, 1\}$.

Exponential error is mainly interesting because it happens to lead to AdaBoost!

It has a few disadvantages compared to cross entropy:

- Exponential error puts more weight on misclassified data points, making it more sensitive to outliers or mislabeled inputs.
- Exponential error cannot be interpreted as a log likelihood function of any simple probabilistic model (we prefer maximum likelihood estimation over ad-hoc methods).
- Exponential error does not generalize to multiclass problems.

However, by replacing the error function, we can derive a variety of boosting algorithms with desired properties.

There has been a lot of research on how to get the good generalization of AdaBoost on multiclass problems and in situations with outliers.

The Viola and Jones face detection algorithm works like this:

- Sweep a small detection window over the image at multiple scales.
- For each candidate face location,
  - Extract a set of features of the pixel values in the detection window. A feature is just an arbitrary (usually but not always linear) mathematical function of the pixels in the detection window.
  - Based on the set of feature values, decide whether the image patch contains a face or not.
- Group together nearby detections and eliminate overlapping detections.

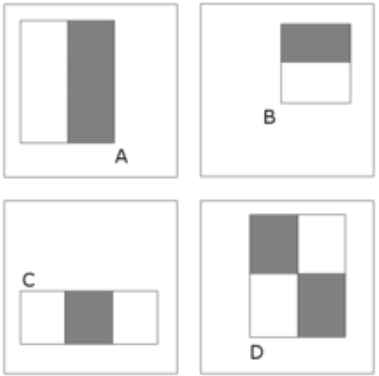
The genius of the method is three main ideas:

- They came up with a type of feature that can be computed extremely fast and is effective in the decision step.
- They used an existing method for learning which features from an enormous set of possible features are effective and how to combine them.
- They came up with a way to quickly discard detection window locations unlikely to contain faces.

Before Viola and Jones: effective but agonizingly slow face detection.

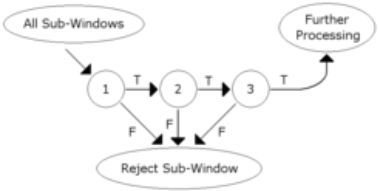
After Viola and Jones: face detection is common in commercial products such as digital cameras.

The features are simple binary convolution filters combined with a threshold:



These are called Haar-like filters and can be computed extremely quickly using an integral image.

The method for discarding detection windows is to arrange the AdaBoost classifiers into a cascade:



Windows unlikely to be faces are quickly rejected, and windows more likely to be faces are given more consideration.

To put the pieces together, we have a learning phase and a runtime phase.

In the learning phase, we collect thousands of images of faces and non faces and run AdaBoost with exhaustive search over the set of possible Haar-like features.

This part took days on a compute cluster in the 2000s. Nowadays, training time is small compared to large CNN models.

In the runtime phase, we simply capture images, sweep the detection window over the image, and apply the feature cascade to determine if the window contains a face or not.

In a final cleanup phase we group nearby detections and eliminate overlapping detections.

Try the "haarcascade" sample in the OpenCV distribution.

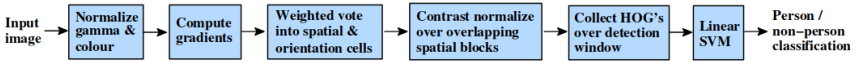
Outline

In 2005, Dalal and Triggs introduced another object detector that we still use today.

Basic idea:

- Apply the multiscale scan window technique
- For each window, extract a SIFT-like descriptor for a rectangular subregion of the image
- Classify the resulting descriptor as object or not object using a linear SVM

More detailed pipeline:



Dalal and Triggs (2005), Fig. 1

Training sets sizes for HOG detectors usually range in the 1000s.

Dalal and Triggs used a dataset of 1239 positive patches and 1218 images without people.

Initial negative set was 12,180 random patches from the negative images.

False positives from exhaustive search with intial detector were added to the negative set for final training.

Sample positives for human detection:

Dalal and Triggs (2005), Fig. 2

We have already discussed how important it is to have a very high classification accuracy when performing object detection.

How to evaluate a detector? There are two criteria:

- Hit rate
- False positive rate

There are many methods to combine hit rate and FP rate into one number.

Dalal and Triggs use the miss rate at 10^{-4} false positives per window as the criterion.

Dalal and Triggs found that detectors work better with color information.

Same process is applied separately to the R, G, and B channels.

Gamma correction is used to increase contrast in darker regions of the image and reduce contrast in brighter regions.

Square root correction performed better than log correction.

Many methods to calculate the gradient were tried.

1-D gradient filters, i.e., $\begin{bmatrix}-1 & 0 & 1\end{bmatrix}$, without any image smoothing worked best.

The gradient is performed over the whole image before the scan window is applied.

Color images are collapsed to a single plane by taking, for each pixel, the gradient with the largest norm over the three color channels. [Clever!]

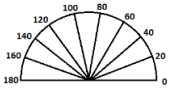
Each pixel's gradient votes for a particular gradient orientation.

Gradients are binned (a histogram is calculated) over small cells.

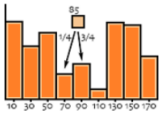
The best voting strategy was to have the vote proportional to the magnitude of the gradient.

Votes are split with bilinear interpolation over neighboring bins to avoid aliasing.

More orientation bins (up to 9) seems to help. Unsigned orientations (0° to 180°) work best for human detection, whereas signed orientations (0° to 360°) work best for some other classes (cars and motorbikes).



9 Bins



Bin centers

Images have different levels of contrast in different regions.

For object classification, usually, the relative magnitude of the gradients is what's important, not the absolute magnitudes.

Local contrast normalization helps us get accurate texture information in both high and low contrast areas.

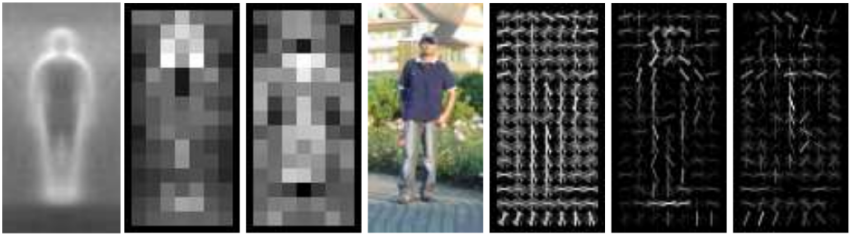
Cells are grouped into blocks (6×6 cells grouped into 3×3 blocks seems to work well), over which the gradient orientation histograms are summed up and normalized.

Normalization turns out to be critical for good performance. Among many methods, L2 normalization ($\mathrm{v} \leftarrow \mathrm{v}/\sqrt{\|\mathrm{v}\|_2^2 + \epsilon^2}$) is found to work as well or better than other methods.

Dalal and Triggs find that a 64×128 detection window with 16 pixels of border around the person in the patch works best.

The SVM is a soft margin linear hyperplane classifier with $c = 0.01$.

The Gaussian RBF kernel gives better performance (3% improvement in the miss rate at 10^{-4} FPPW) but requires a much higher runtime [recall why RBF increases runtime so much?].



Dalal and Triggs (2005), Fig. 6

Visualization of the HOG descriptor is useful. (a) Average gradient over training examples. (b) Maximum positive SVM weight in each block of the descriptor. (c) Maximum negative weight in each block. (d) Sample test image. (e) HOG descriptor of image in (d). (f,g) The HOG descriptor weighted by positive and negataive SVM weights.

The HOG paper is a beautiful example of how to carefully engineer a machine vision system. Everyone should read it!

HOG detectors have competitive performance even today.

See the HOG implementation in OpenCV, and try the "peopledetect" sample.

HOG represents a pinnacle in the development of feature-based methods for object detection. HOG sadly may have been the last great feature-based method in machine vision!

Remember that in machine learning, rather than hand-crafting classification rules, we let the data decide what the parameters should be.

To further improve classifier performance in difficult, cluttered environments, we need to return to Viola and Jones' idea of letting the data decide not only the parameters of the final classifier, but also to decide what features are best for a particular task.

This, of course, leads us to neural networks and end-to-end learning.

Outline

Note: this material on CNNs is repeated from Recent Trends in Machine Learning, for reference.

Convolutional neural networks (CNNs) are good for data with a known grid-like topology:

- Time series form 1-D grids.
- Images form 2-D grids.

Mathematically: convolution is an operation on two functions with a real-valued (possibly vector-valued) argument.

Continuous convolution for smoothing (Goodfellow et al., 2016)

We might use convolution for smoothing a 1-D function such as continuous noisy measurements $x(t)$ of the linear position of a spaceship:

$$s(t) = \int x(a)w(t-a)da$$

$$s(t) = (x * w)(t)$$

$w(\cdot)$ in the smoothing example:

- Should be a probability density function to make it a weighted average.
- Should be 0 for negative arguments (the future)

There are many other applications of continuous convolution.

Now we move to discrete convolution.
$x(t)$ is assumed to be a discrete sample from some underlying continuous function taken at regular spatial or temporal intervals.

We'll call $x(t)$ the input and $w(t)$ the kernel.

The output is usually called a feature map.

The continuous integral becomes a discrete sum:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

In a ML application, the input is an arbitrary multidimensional array of data and the kernel is a multidimensional array of parameters adapted by the learning algorithm.

Multidimensional arrays are called tensors.

The theoretically infinite functions are assumed 0 everywhere but in the finite set of points we have stored.

Example: a two-dimensional image $I(\cdot,\cdot)$ would best be processed by a two-dimensional kernel $K(\cdot,\cdot)$:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

Convolution is commutative, so we can equivalently write

$$S(i,j) = (I * K)(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n).$$

This version is easier to implement, as the loop is over the smaller number of non-zero values in $K(\cdot,\cdot)$.

Note that in both $I * K$ and $K * I$, as the index into the input increases, the input into the kernel decreases, and vice versa.

We have flipped the kernel.

Flipping the kernel is necessary mathematically for commutativity, but is not necessary for the actual information processing. Therefore, we may use cross-correlation

$$S(i,j) = (K \star I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n),$$

in which we don't flip the kernel. You will often see cross-correlation called convolution in machine learning libraries.

In this class, when we say "convolution" we will normally mean cross correlation!

2D convolution operations give us the linear response of a 2D filter applied to the pixels in a local region of an image.

See any of many examples on YouTube, such as
`https://www.youtube.com/watch?v=_iZ3Q7VXiGI` !

The simplest example is edge detection using, for example, Sobel filters:

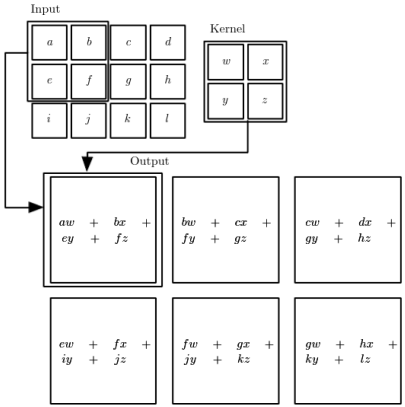
$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

A convolution may apply to the input or a feature map from a previous layer.

Typically, convolutions apply to all of the feature maps in the previous layer: this is called convolution over volume.

- For the input: three maps (R, G, and B) or one map (grayscale).
- For an inner layer: the number of kernels. This would be, e.g., 96 for AlexNet's second convolutional layer, except that AlexNet splits into two separate hierarchies so it is 48 for each stream.

Example below. Note that we only use the valid parts of the convolution where the kernel fully overlaps the valid part of the input.



Goodfellow et al. (2016), Figure 9.1

There are three main important ideas behind the success of convolutional neural networks:

- Sparse interactions
- Parameter sharing
- Equivariant representations

(Aside: a benefit of convolutions is that they can be applied to an input of variable size.)

Sparse interactions:

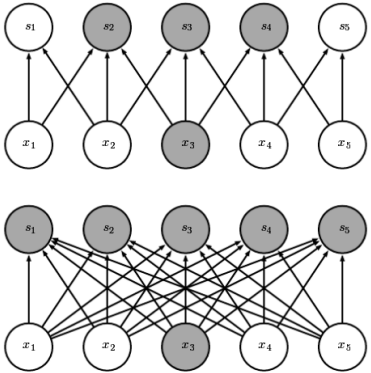
- A fully connected layer has dense interactions (every unit interacts with every unit in the previous layer)
- A convolutional layer has sparse interactions (each unit in the output feature map interacts only with a few elements of the input, via the small kernel.

Fewer parameters means lower memory requirements and higher statistical efficiency.

The specific organization of a convolution also means it can be implemented efficiently.

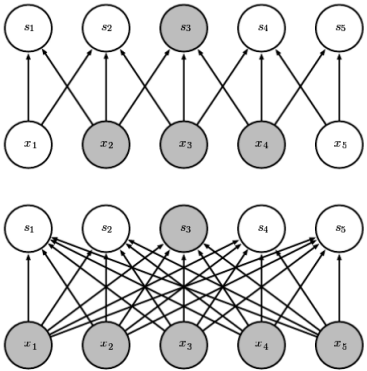
Dense connections from m inputs to n outputs requires $O(mn)$ time. With a kernel of size k, we only need $O(kn)$ time.

Sparse interaction example. Unit x_3 only affects units s_2, s_3, and s_4 in the output feature map. Compare to dense interaction below.



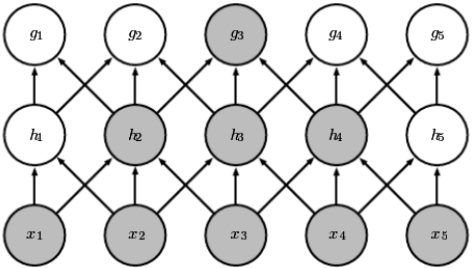
Goodfellow et al. (2016), Figure 9.2

Looking from above, unit s_3 in the output feature map has a receptive field including x_2, x_3, and x_4.



Goodfellow et al. (2016), Figure 9.3

When convolutional layers are formed into hierarchies, a unit in a deep layer (g_3) can be indirectly connected to all or most of the input.



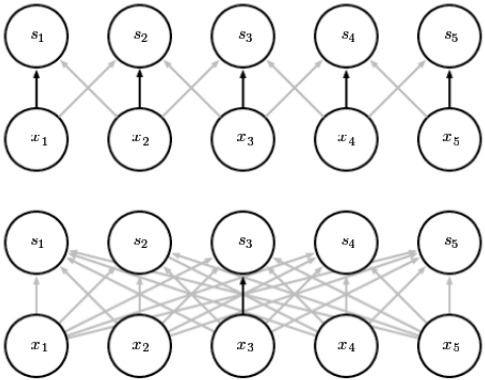
Goodfellow et al. (2016), Figure 9.4

Parameter sharing:

- Sharing means using the same parameter for more than one interaction in a model.
- Shared parameters are also often called tied weights.
- In a convolution, each weight in the kernel is reused at every position in the input.

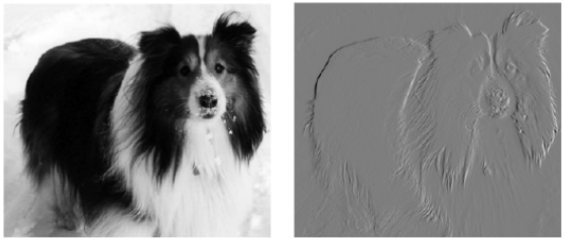
Parameter sharing increases statistical efficiency.

With parameter sharing (top), one parameter is used many times. Without parameter sharing, one parameter is used only once.



Goodfellow et al. (2016), Figure 9.5

The representational power of convolution with shared parameters is clear with the following example of vertical edge detection using a 2-element kernel (-1,1):



Goodfellow et al. (2016), Figure 9.6

Equivariance:

- A function $f(x)$ is equivariant to function g if $f(g(x)) = g(f(x))$.
- Convolution is equivariant to translation.
- If f is a convolution operation and g is a translation operation, we can see that the convolution of a translated version of input x is the same as the translation of the convolution of f and the input x.

How is equivariance useful?

For time series: we get the same response to the same event, regardless of when the event occurs.

For images: we get the same response to a local pattern, regardless of where in the image it occurs.

Note that convolution is NOT equivariant to scale, rotation, etc.

One of the insights of the CNN is to perform convolutions hierarchically.

However, a hierarchy of purely linear transformations would ultimately be equivalent to a single linear transformation, which would not give powerful pattern matching capabilities.

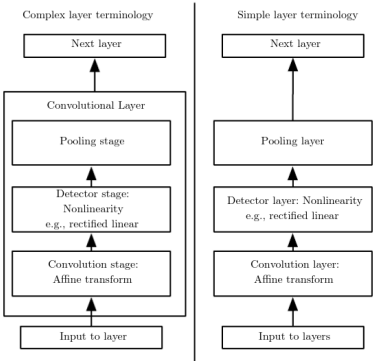
Generally, then, the result of one convolution in the hierarchy should be transformed by a nonlinearity.

ReLU is the most popular nonlinear activation function.

Hyperbolic tangent and the logistic sigmoid are also possible but lead to slower learning, according to Krizhevsky et al. (AlexNet).

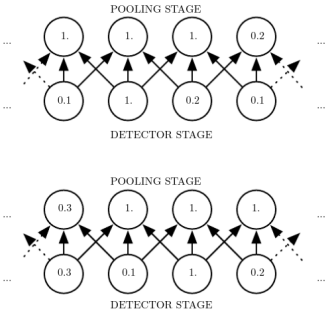
The resulting feature map may possibly be downscaled by a pooling operation before it is convolved with higher-level filters.

The structure of a CNN usually contains several "macro" layers consisting of a convolution, a nonlinearity especially ReLU, then pooling. Sometimes a "layer" means all three operations, and sometimes each operation is treated separately.



Goodfellow et al. (2016), Figure 9.7

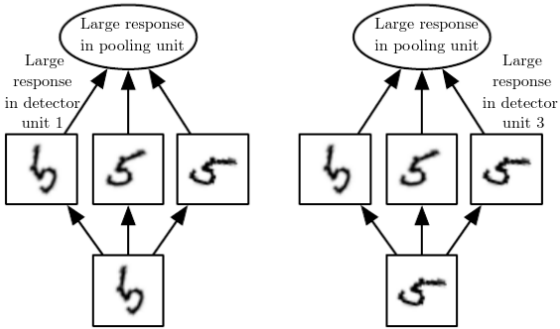
Pooling makes the output feature map approximately invariant to small translations of the input.



Goodfellow et al. (2016), Figure 9.8

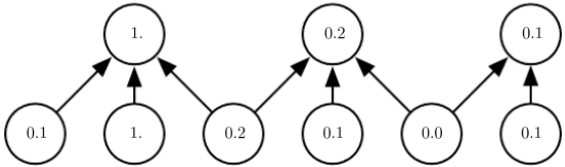
This is good when we want to know whether a feature is in the input, without knowing precisely where it is.

Pooling over features enables invariance to more complex transformations of the input, such as rotation:



Goodfellow et al. (2016), Figure 9.9

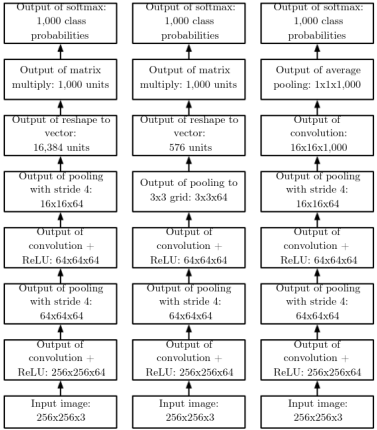
Usually, pooling is combined with downsampling, which reduces the computational and statistical burden on the next layer.



Goodfellow et al. (2016), Figure 9.10

Some sample CNN architectures, as examples. Note that practical networks are deeper, more variable, and have branches.

Generally, the kinds of processing we want to do on spatial and temporal data fit the convolve + pool processing approach, but it may not always be so. Convolution and pooling may cause underfitting.



Goodfellow et al. (2016), Figure 9.11

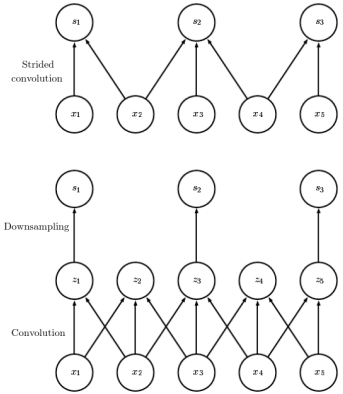
In general, the input, kernel, and output are all tensors of arbitrary number of dimensions.

Images: 2D data with 3 channels (red, green, blue), which may be combined into mini-batches, giving a 4D tensor as the input.

The kernel applied to these data might be a 4D tensor $\mathtt{K}_{i,j,k,l}$ giving strength of connection between input channel i, output channel j, and offset (k, l) between the output and input unit. Note that every feature map j connects with every channel i in the input.

The output would then be a 3D tensor $\mathtt{Z}_{i,j,k}$, where i represents the output channel or feature map and (j, k) indicates the spatial location.

We may also downsample during the convolution operation, using a stride not equal to 1. Example stride of 2:



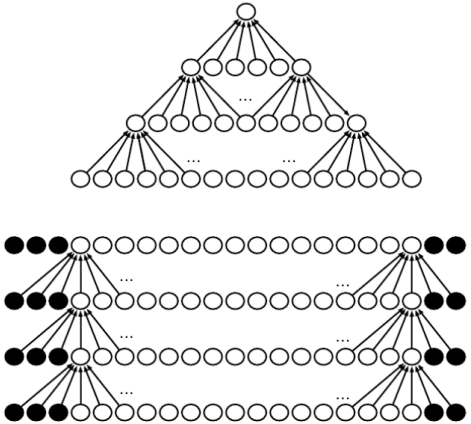
Goodfellow et al. (2016), Figure 9.12

Stride is important:

- Typically, the convolution operation is applied at every pixel of the input (stride $= 1$).
- When spatial resolution is less important or neighboring receptive fields overlap significantly, we may skip some pixels of the input (stride > 1).
- Most architectures use a stride of 1 for 3×3 convolutions and a stride of 1–3 for 5×5 convolution.
- The trend: smaller kernels and more layers $\rightarrow$ small strides (11×11 convolutions as in AlexNet are not often seen).

We may also pad the image so that the valid convolution at each layer has the same size as the input to the layer:

Usually, the padding is 0.



Goodfellow et al. (2016), Figure 9.13

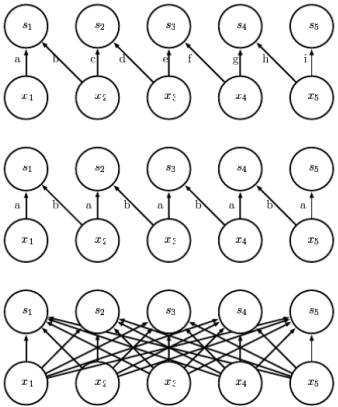
Padding is important:

- Without padding, the border would shrink after each convolution, and information at the image border would be lost.
- In most cases, we should add padding necessary so that the output feature map has the same size as the input feature map when the stride is 1.
- The most common choice for padding seems to be 0-padding.
- Matt likes copy-border padding, but most libraries do not implement it.
- Most libraries do implement "reflect" padding which seems to work well (and is better than zero-padding).

In unshared convolution, we perform the same dot product of a set of weights over a local region in the input, but do not use a shared kernel.

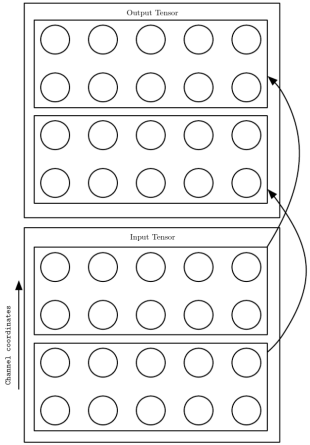
This makes sense when we need local sparse computation but have no reason to apply the same operation throughout the input.

Comparison of local connections, convolution, and full connection:



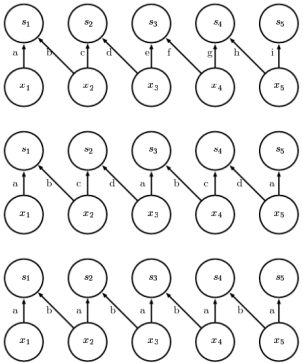
Goodfellow et al. (2016), Figure 9.14

We may also want separate kernels for each input channel:



Goodfellow et al. (2016), Figure 9.15

Another variation is tiled convolution in which we combine the idea of having separate weights for neighboring positions in the feature map but also sharing parameters:



Goodfellow et al. (2016), Figure 9.16

Transpose convolution means multiplication by the transpose of the matrix defined by convolution. It is used for

- Backpropagating error derivatives through a convolutional layer
- Reconstructing input units from hidden units in an autoencoder

Suppose we have a convolution tensor $\mathtt{K}$ applied to multichannel tensor $\mathtt{V}$ with stride s: $c(\mathtt{K}, \mathtt{V}, s)$.

Suppose the loss function is $\mathtt{J}(\mathtt{V}, \mathtt{K})$.

Forward propagation gives us $\mathtt{Z} = c(\mathtt{K}, \mathtt{V}, s)$.

$\mathtt{Z}$ is propagated forward, then during backpropagation, we receive tensor $\mathtt{G}$ where

$$\mathtt{G}_{i,j,k} = \frac{\partial}{\partial \mathtt{Z}_{i,j,k}} \mathtt{J}(\mathtt{V}, \mathtt{K}).$$

The derivative of $\mathtt{J}$ with respect to weight i, j, k, l in $\mathtt{K}$ is

$$g(\mathtt{G}, \mathtt{V}, s)_{i,j,k,l} = \frac{\partial}{\partial \mathtt{K}_{i,j,k,l}} J(\mathtt{V}, \mathtt{K}) = \sum_{m,n} \mathtt{G}_{i,m,n} \mathtt{V}_{j,(m-1)\times s+k,(n-1)\times s+l}.$$

The deltas to be backpropgated to previous layers are

$$h(\mathtt{K}, \mathtt{G}, s)_{i,j,k} = \frac{\partial}{\partial \mathtt{V}_{i,j,k}} J(\mathtt{V}, \mathtt{K})$$

$$= \sum_{\substack{l,m \\ \text{s.t.}(l-1)\times s+m=j}} \sum_{\substack{n,p \\ \text{s.t.}(n-1)\times s+p=k}} \sum_{q} \mathtt{K}_{q,i,m,p} \mathtt{G}_{q,l,n}.$$

How the bias term is done varies.

For locally connected layers, each unit would have its own bias.

For convolutional layers, usually, each channel of the output has a separate bias shared across all locations.

Sometimes, each element in the output will have its own bias. This is useful for example when we use 0 padding and border elements receive less input than interior elements.

Besides classification, convolutional networks can also be used to produce a structured output, itself a multidimensional tensor such as an image.

Besides images, the input data could be a single channel audio waveform, or a sequence of feature vectors corresponding to samples over time, or a 3D volume, or a color video,

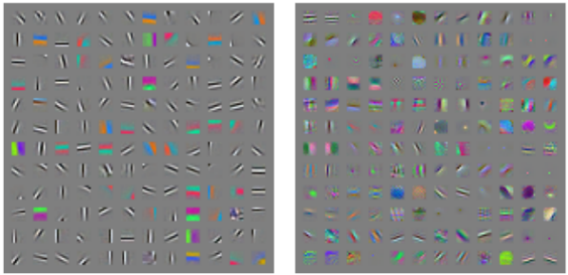
To make convolution efficient, we may use parallel hardware (GPUs).

In some cases, kernels may be separable, e.g., a 2D convolution is the composition of two 1D convolutions in the different dimensions.

How to obtain convolution kernels without expensive backpropagation?

- Random kernels
- Design by hand (e.g., edge detectors or Gabor filters)
- Learn with an unsupervised criterion, e.g., k-means clustering of image patches in the training set.

There is an interesting correspondence between low-level processing in well-trained convolutional neural networks for visual tasks and the kinds of visual feature extractors known to be present in the mammalian visual system:



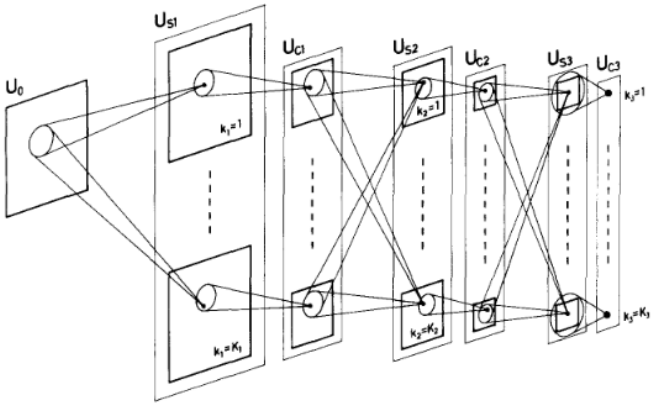
Goodfellow et al. (2016), Figure 9.19

Outline

CNNs for image classification

Origins

The story of the CNN begins in the 1980s with Fukushima's Neocognitron, a hierarchical nerual network designed in primitive mimicry of the hierarchical processing in the primate visual cortex.



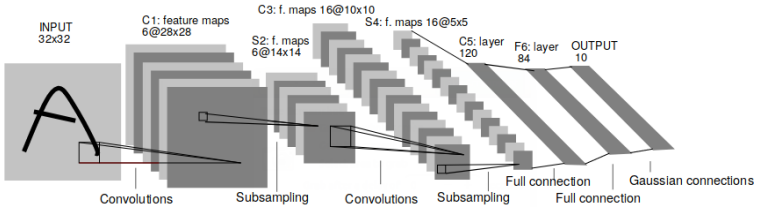
CNNs for image classification

Origins

Geoff Hinton was one of the rediscoverers of backpropagation in 1986.

Hinton's postdoc Yann LeCun went on to create the first practical modern convolutional neural networks for OCR at AT&T Research.[1]

LeCun and colleagues' LeNet-5 (1998) had the world's best performance at recognizing handwritten digits for several years.



LeCun et al. (1998), Fig. 2

[1] Today, Yann LeCun is chief AI scientist at Facebook.

The 2000s were a golden era for feature-based methods like HOG.

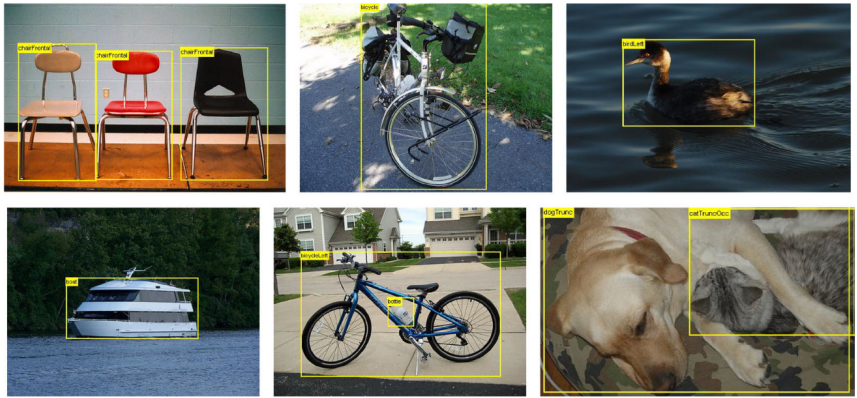
Some research continued, but most vision researchers ignored CNNs.

The main development pushing work forward was the emergence of standardized large-scale datasets.

2006: the PASCAL Visual Object Classification (VOC) challenge started with 20 object categories and ran with more difficult datasets each year until 2012.

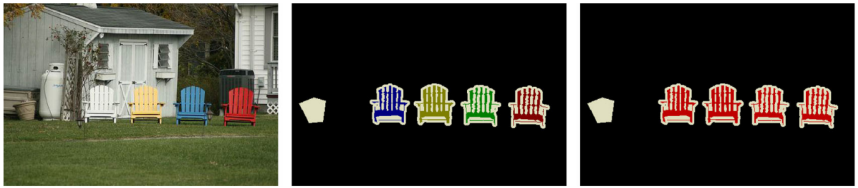
"Simple" feature based methods like HOG could not cope with VOC.

Sample VOC object detection and classification images:

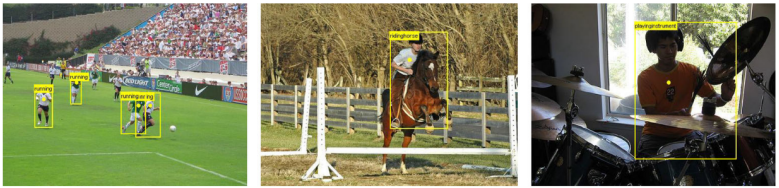


Everingham et al. (2015), Fig. 1a

Sample VOC segmentation and action classification images:



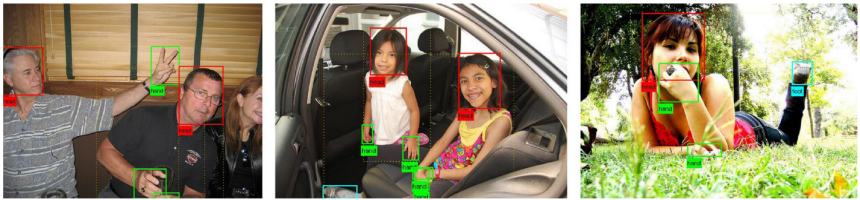
(b) Segmentation



(c) Action classification

Everingham et al. (2015), Fig. 1b-c

Sample VOC person layout images:



Everingham et al. (2015), Fig. 1d

The yearly VOC challenges helped push progress forward, but in retrospect it is clear that the dataset was too small.

Another competition, ImageNet, had even greater influence than VOC.

2007: Fei-Fei Li, then at Princeton and later at Stanford, undertook a new effort to create a visual version of George Miller's WordNet, to be called ImageNet.

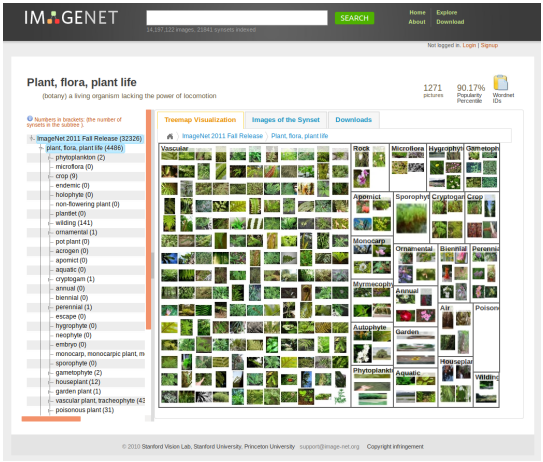
Li failed to get much funding for the project in the beginning but found that Amazon Mechanical Turk could be used to get humans to label images relatively cheaply.

2009: ImageNet was released at CVPR in a poster session then joined forces with PASCAL VOC for a 2010 competition: the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC).

Eventually, the dataset had over 15 million images over 22,000 categories.

The 2010 contest dataset comprised 1.2 million images over 1000 categories.

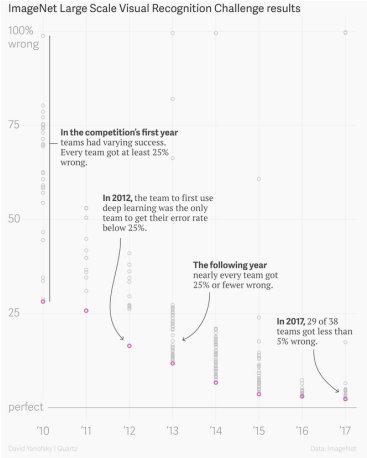
Samples from ImageNet:



http://image-net.org/explore

CNNs for image classification

Origins



https://qz.com/1034972/the-data-that-\
changed-the-direction-of-ai-research-\
and-possibly-the-world/

2010–2011: ImageNet competitors all had error rates over 25%.

2012: Krizhevsky, Sutskever, and Hinton submit AlexNet, sparking today's explosion of interest in AI and deep learning.

CNNs for image classification

AlexNet (2012)

Krizhevsky et al. begin with some provacative points:

- Nature is diverse. We need extremely large datasets if we hope to learn that diversity.
- Learning from millions of images requires models with large capacity.
- CNN capacity can be controlled by varying their depth and breadth.
- The number of parameters in a CNN is smaller than in similar-sized fully connected models.
- CNNs decrease the number of parameters by making assumptions that seem to be correct: relevant features' statistics are stationary, and dependencies between pixels are mostly local.

These factors give us hope that CNNs may have the capacity to learn large datasets with relatively few parameters.[2]

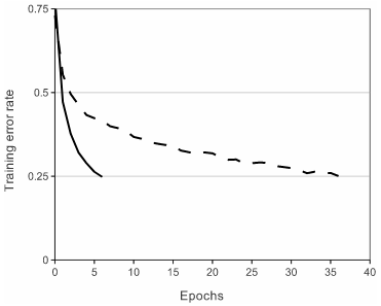
[2] Recall that fewer parameters generally means lower VC dimension which in turn generally means better generalization.

The authors trained what as of 2012 was the largest CNN ever, with 60 million parameters.

Training required a highly efficient implementation of the learning and runtime operations on GPUs with C++ and CUDA.

Training time was 5–6 days on two GTX 580 3GB GPUs.

Current version of the toolkit the authors built is available as open source:
`https://code.google.com/archive/p/cuda-convnet2/`



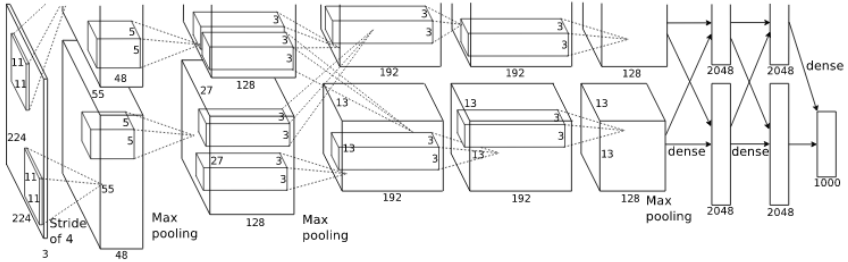
Krizhevsky et al. (2012), Fig. 1

The authors were among the first to exploit the benefits of ReLU over other nonlinear activation functions.

ReLU models learn much faster than tanh models. tanh "saturates" at large absolute values (learning depends on the slope of the activation function).

Training a 4-layer network on CIFAR 10: solid line shows error rate with ReLU, dashed line shows error rates with tanh.

AlexNet begins with a preprocessing step in which the input image is scaled to 256 pixels in the shortest dimension then is cropped to 256×256.



Krizhevsky, Sutskever, and Hinton, (2012), Fig. 2

Five convolutional layers with ReLU activations (Nair and Hinton, 2010) and max pooling layers are followed by three fully-connected layers.

AlexNet layers:

- $224 \times 224 \times 3$ input
- Convolution with 96 kernels of size $11 \times 11 \times 3$ with a stride of 4 pixels
- ReLU + local response normalization + overlapping max pooling
- Convolution with 256 kernels of size $5 \times 5 \times 48$ with a stride of 1.
- ReLU + local response normalization + overlapping max pooling
- 384 kernels of size $3 \times 3 \times 256$ with a stride of 1
- ReLU only (no normalization or pooling)
- 384 kernels of size $3 \times 3 \times 192$
- ReLU only
- 256 kernels of size $3 \times 3 \times 192$.
- ReLU only
- 4096 fully connected units
- 4096 fully connected units
- 1000 fully connected softmax units

After ReLU, the next most important technique used is local response normalization, which reduces errors by more than 1%.

Non-saturating activation functions means we can have very large activations for some inputs.

Response normalization reduces large activations while preserving relative relationships between different feature maps.

Another term for this is local inhibition, a property seen in real neural circuits.

Letting $a^i_{x,y}$ represent the ReLU activation feature map i at location (x, y), the normalized response is

$$b^i_{x,y} = a^i_{x,y} \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} \left(a^j_{x,y}\right)^2 \right)^{\beta}$$

We are normalizing each unit relative to its n neighboring features.

Another technique is overlapping pooling.

Before AlexNet, pooling usually used a stride equal to the width of the pooling region, so that neighboring pooled units did not have overlapping receptive fields.

The authors find, however, that overlapping pooling is effective. They use a stride of 2 and a pooling region of size 3x3.

CNNs for image classification

AlexNet (2012)

How to avoid overfitting when we have 60 million parameters?

For images, data augmentation using various image transformations makes sense.

AlexNet authors begin with a 256×256 image then sample 224×224 patches from the original.

- Training time: 2048 random patches including translation, horizontal reflections, and global random intensity transformations per image.
- Test time: four corner patches plus the center patch are processed, and the output layer is averaged over the 5 samples.

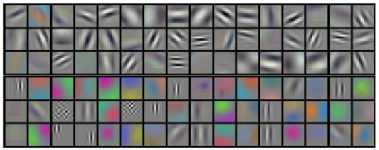
The second trick is dropout at the (first two) fully connected layers:

- Training time: each output in the feature map is set to 0 with probability 0.5. Zeroed outputs do not get any backpropagated error.
- Test time: all units are used but output is multipled by 0.5.

Training parameters:

- Stochastic gradient descent
- Batch size 128 examples
- Momentum 0.9
- Weight decay 0.0005
- Weights initialized with samples from $\mathcal{N}(0, 0.01)$
- Biases initialized to 1 for most layers (to place ReLU in the positive region) and 0 for other layers (first and third convolutional layer).

The first convolutional layer learns representations reminiscent of neurons in visual cortex:



Krizhevsky et al. (2012), Fig. 3

This is after 90 epochs through the 1.2 million images in ImageNet.

The result: top-5 error rate dropped from 26% (2011) to 15.3%.

In the 2013 ILSVRC, a “tweaked” version of AlexNet reduced the top-5 error rate to 12%.

CNNs for image classification

GoogLeNet (2014)

In the 2014 ILSVRC, Google's entry achieved a further huge improvement to a 6.7% top-5 error rate.

Principles: smaller convolutions, more layers, inception modules.

AlexNet's 60 million parameters reduced to 4 million.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015), Going deeper with convolutions. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.

Szegedy et al. (2014), Fig. 3

Before Inception, accuracy was achieved by larger networks with more feature maps and more parameters.

Overfitting is avoided by aggressive data augmentation and dropout.

Inception idea: can we have deeper, wider networks with a fixed computational budget?

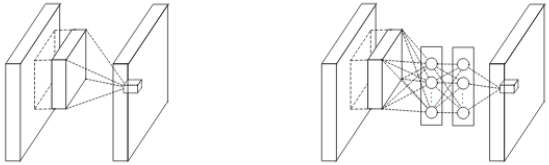
Goal: 1.5 billion multiply-adds at inference time.

This was achieved with a 22-layer (27 including pooling layers) model with a modular structure.

The module idea was inspired by "Network in Network."

In 2013, Lin, Chen, and Yan at NUS had introduced the concept of Network-In-Network:

- In place of simple convolutions, local operations are performed by small multilayer perceptrons.
- The entire module is then scanned over the input like a convolution to produce a new feature map.



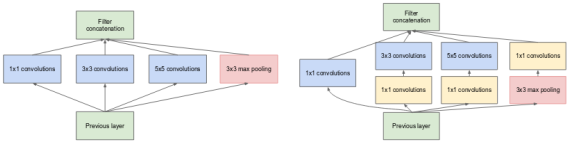
(a) Linear convolution layer (b) Mlpconv layer

Lin, Chen, and Yan (2013), Fig. 1

Inception modules simplify the NIN concept, replacing the MLP with a single $1\times1\times D$ convolution followed by ReLU, which can be implemented with standard CNN tools.

The 1×1 convolutions also aim to capture some of the theoretical work suggesting that extracting and combining sparse clusters of features over the image is optimal.

An inception module thus combines 1×1, 3×3, and 5×5 convolutions all feeding a single aggregating feature map.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Szegedy et al. (2014), Fig. 2

1x1 convolutions occur both before larger 3x3 and 5x5 convolutions, and after them.

Before: a reduction step that reduces the number of feature maps the 3x3 or 5x5 operates over, making them more efficient.

After: a project step where redundancy is removed and output dimensionality is reduced.

Besides an approximation to network-in-network, the architecture is intended to approximate theoretically optimal construction of large, sparse, locally connected layers that progressively cluster correlated features at previous layers.

Proportion of 3x3 and 5x5 convolutions increases at later layers.

Pooling layers are used, but unpooled features are also propagated, to get both spatial accuracy and translation invariance.

Auxiliary classifiers are used at intermediate layers to increase their discrimination ability.

Training was CPU-only, with SGD, Polyak averaging, multiple models at prediction time, and aggressive data augmentation.

The model was also used for detection in ILSVRC 2014, applying selective search for region proposals and GoogLeNet to classify those regions, without bounding box regression.

GoogLeNet Image classification setup:

- 7 different models with different training pattern sampling.
- Test images are scaled to four sizes (256, 288, 320, and 352).
- For each scaled image, the left, center, and right or top, center, and bottom squares are taken.
- For each such image, 5 224×224 crops (4 corners plus center) and the entire region scaled to 224×224 are taken.
- For each crop, we take the original and horizontally flipped image.
- Total test images per input: $4 \times 3 \times 6 \times 2 = 144$.

After 2014, research on image classification network continued at a fast pace.

Several labs continued to improve their models to perform better at ILSVRC.

Szegedy and colleagues increased GoogLeNet's size to 5 billion multiply-adds, trying to use that budget as efficiently as possible.

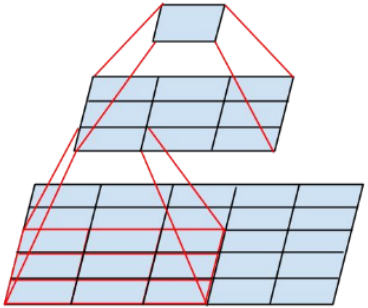
Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016), Rethinking the Inception architecture for computer vision, *CVPR*.

Some principles:

- Avoid information bottlenecks early in processing. Decrease dimensionality gradually as we move deeper.
- Keep dimensionality high for local processing.
- Reduce dimensionality before performing spatial aggregation.
- Balance the width and the depth of the network. Increased capacity should be achieved by increasing both depth and width.

The authors discuss a variety of ways of reducing computation without reducing representational capacity.

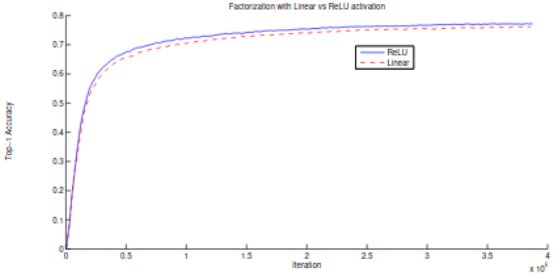
Example: replacing a 5x5 convolution with a hierarchy of two 3x3s:



Szegedy et al. (2016), Figure 1

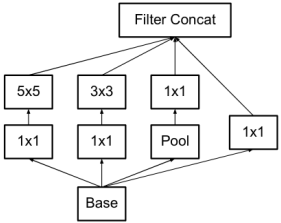
When we factorize convolutions in this way, should we apply ReLU to every layer or only at the end?

Experiments show that ReLU at every step is better than linear activations in the factored layers:

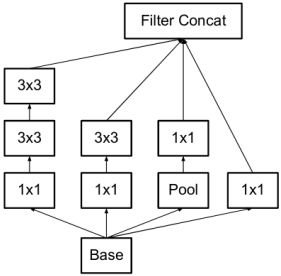


Szegedy et al. (2016), Figure 2

The 5x5 factorization gives a new structure for the basic Inception module:



Szegedy et al. (2016), Figure 4

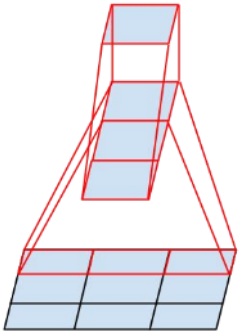


Szegedy et al. (2016), Figure 5

Another way of factorizing: replacing a 3x3 with a 3x1 followed by a 1x3.

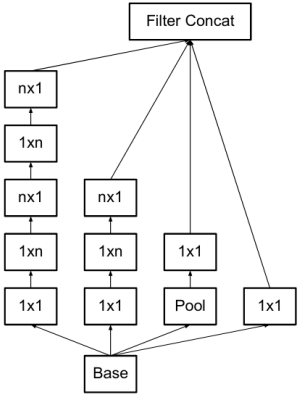
This process can go on; any nxn convolution can be factored into a nx1 followed by a 1xn.

The authors find this is not very useful in early layers, but is effective for medium-sized grids of 12-20 units wide.



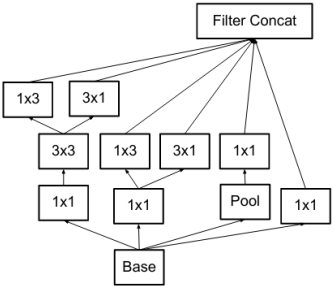
Szegedy et al. (2016), Figure 3

Inception module for medium-sized grids:



Szegedy et al. (2016), Figure 6

Inception module for coarsest grids:



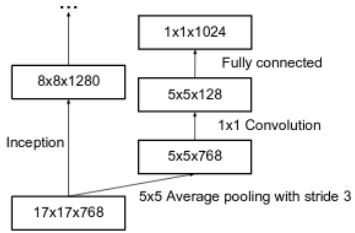
Szegedy et al. (2016), Figure 7

The side classifiers from GoogLeNet turn out not to be as effective as first thought.

Removing one of the two side classifiers does not adversely affect performance.

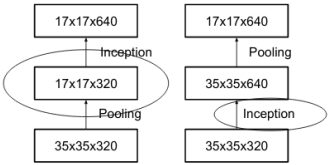
The side classifier seems to work as a regularizer rather than helping to create more discriminative features in early layers.

Auxiliary classifier on top of the last 17x17 feature map:



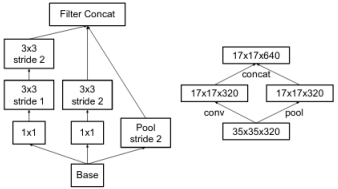
Szegedy et al. (2016), Figure 8

There are multiple ways of reducing grid size. The left structure introduces a bottleneck. The right structure avoids the bottleneck but introduces more computation.



Szegedy et al. (2016), Figure 9

Alternative: perform pooling with stride 2 and convolution with stride 2 in parallel:



Szegedy et al. (2016), Figure 10

Putting all these ideas together resulted in Inception-v2, a 42-layer extension of GoogLeNet.

Additional features leading to Inception-v3:

- RMSprop optimizer
- Label smoothing: rather than one-hot ground truth, we mix the one-hot distribution with the prior distribution over the classes.
- Batch normalization of FC layers and conv layers in the side network.

Result was the highest single-crop top-1 accuracy on ILSVRC to date.

CNNs for image classification

VGG (2014)

The 2nd place entry in 2014 was VGG (Simonyan and Zisserman, 2014).

Important features:

- 3×3 filters only
- 16–19 layers
- Otherwise similar to AlexNet
- 138 million parameters
- Tested on multiple crops through additional convolutional steps rather than averaging multiple crops

We learn that a deeper network with smaller convolutions is better than a shallower network with larger convolutions.

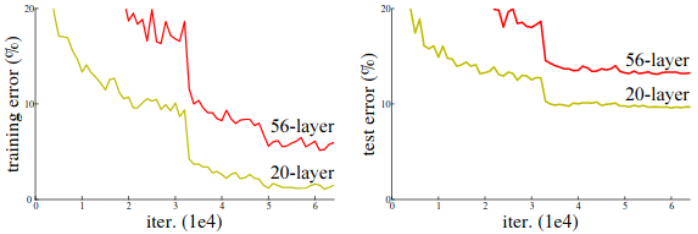
The 2015 ILSRVC winner was ResNet from Microsoft Research.[3]

He, K., Zhang, X., Ren, S., and Sun, J. (2016), Deep Residual Learning for Image Recognition, In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 770–778.

ResNet pushes the notion that "more depth is better" to the extreme.

The problem faced by very deep networks is degredation: though adding more layers improves training error to a point, eventually, training error starts to increase.

[3] Note, though, that Baidu had an entry that beat ResNet, but the entry was disqualified for cheating.



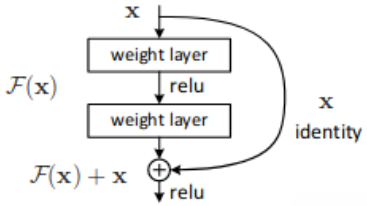
He et al. (2016), Fig. 1

As we are talking about training error, the degredation is not overfitting.

A deeper model should be at least as good as its shallow cousin (think about constructing a deep network from a shallow one by adding identity mappings).

Degredation is due to the vanishing integrity of the training signal as the network gets deeper.

To overcome degredataion in very deep networks, ResNet uses the concept of residual learning:



He et al. (2016), Fig. 2

To learn a mapping $\mathcal{H}(\mathrm{x})$, we let intermediate layers learn another mapping $\mathcal{F}(\mathrm{x}) = \mathcal{H}(\mathrm{x}) - \mathrm{x}$ then compute $\mathcal{H}(\mathrm{x})$ at the output.

Residual learning can be implemented with shortcut connections that add the input to the output of the subnetwork.

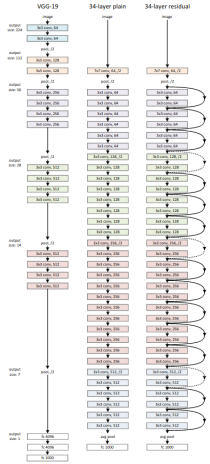
When a subnetwork changes the dimensionality of input x, then a dimensionality changing mapping is used instead of the identity mapping.

He et al. demonstrate that

- Very deep networks without shortcut connections fail to learn the training set as well as similar networks with shortcut connections.
- A 152-layer network with shortcut connections can learn on ImageNet and CIFAR and won ILSVRC 2015 (3.56% top-5 error).

CNNs for image classification

ResNet (2015)

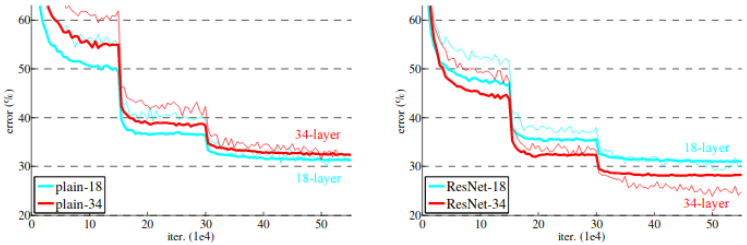


He et al. (2016), Fig. 3

Baseline model (middle) is a 34-layer network

- Mostly 3×3 convolutions
- Equal-size mappings always have the same number of filters
- Mappings that downsample do so by half, with double the number of filters

Shortcut connections in residual version (right) are performed by 1×1 convolutions with stride of 2.



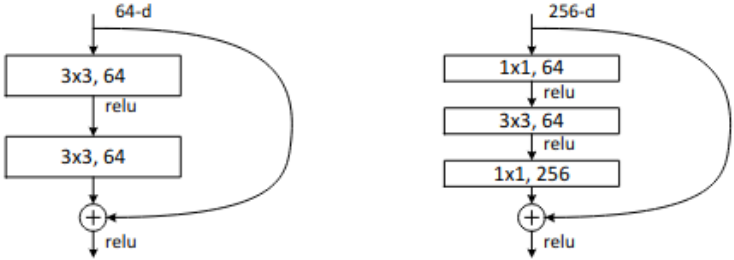
He et al. (2016), Fig. 4

Left: plain networks with 18 or 34 layers. Right: residual networks with 18 or 34 layers.

Residual layers give better convergence for the small network and better accuracy for the larger network. Only the larger residual model pushes training error lower than validation error.

Identity or projection? Experiments show not much impact of either choice.

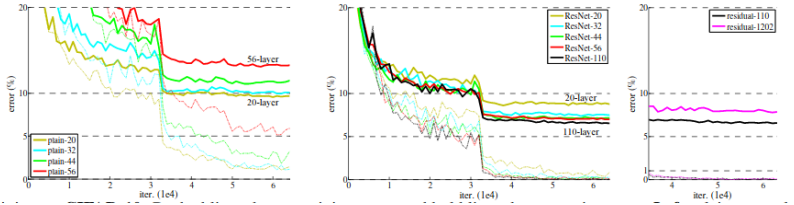
The authors decided to use identity for same-size mappings and projection for decreased-size mappings.



He et al. (2016), Fig. 5

Ordinary residual block (left) and bottleneck residual block (right).

Bottlenecks do not seem to hurt performance and are more economical in terms of calculations and number of parameters, so are used in the 152-layer ResNet.



He et al. (2016), Fig. 6

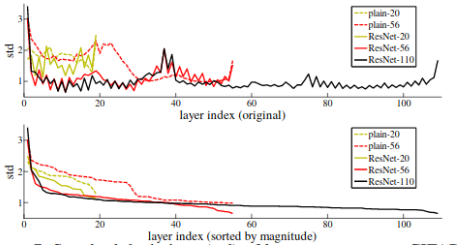
CIFAR results. Left: plain networks perform worse with more layers, both training and test.

Middle: residual networks perform better with more layers, both training and test.

Right: extremely large residual networks with more than 1000 layers learn well but exhibit overfitting.

CNNs for image classification

ResNet (2015)



He et al. (2016), Fig. 7

Top: standard deviation of layer output responses after BN but before skip connection adding and ReLU.

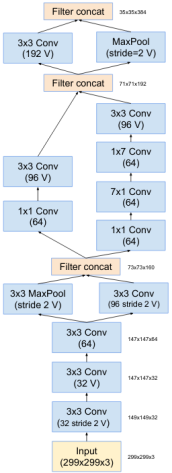
Bottom: same data, sorted.

Residual layers are less active than plain layers.

The Inception group, inspired by the success of ResNet in ILSVRC 2015 and other competitions, considered whether residual connections can improve Inception.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2017), Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *Association for the Advancement of Artificial Intelligence Conference on AI (AAAI)*.

A version of Inception-ResNet achieved a 3.08% top-5 error rate on ILSVRC 2016 (but was beaten by Trimps-Soushen with 2.99%).

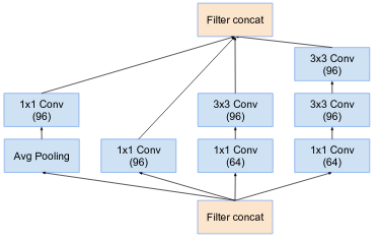


Szegedy et at. (2017), Fig. 3

Stem network used at the input of Inception v4 and Inception-ResNet v2.

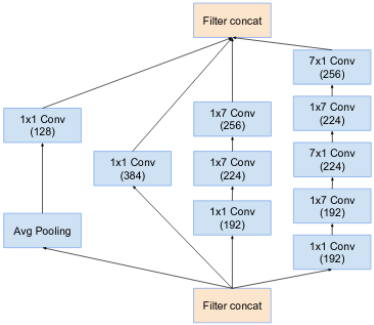
No "V" means the input is same-padded so that output is same size as input.

"V" means valid only.



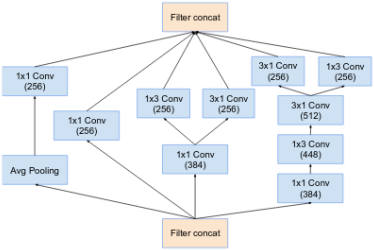
Szegedy et at. (2017), Fig. 4

35×35 Inception module for Inception v4 (Inception A).



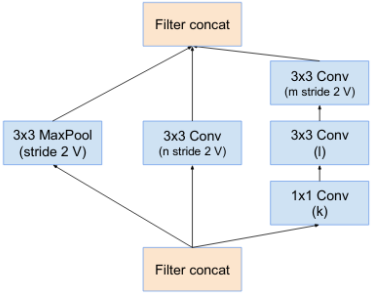
Szegedy et at. (2017), Fig. 5

17×17 Inception module for Inception v4 (Inception B).



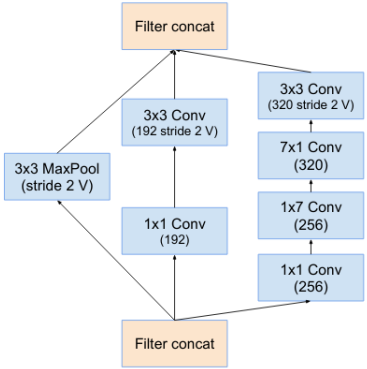
Szegedy et at. (2017), Fig. 6

8×8 Inception module for Inception v4 (Inception C).



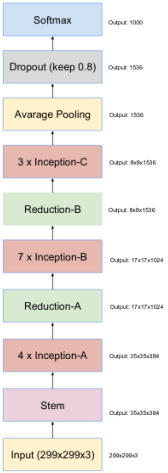
Szegedy et at. (2017), Fig. 7

Module for reduction from 35×35 to 17 × 17 (Reduction-A).



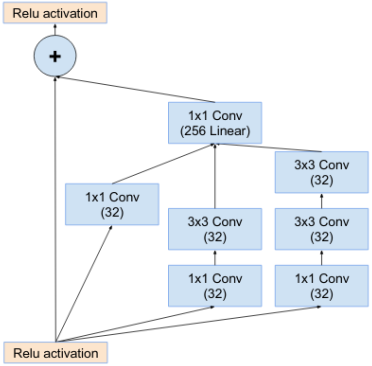
Szegedy et at. (2017), Fig. 8

Module for reduction from 17×17 to 8×8 (Reduction-B).



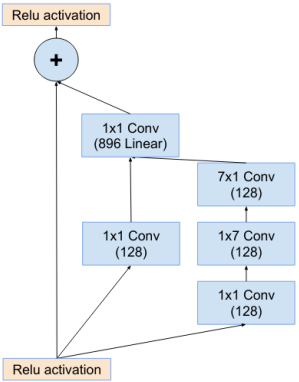
Szegedy et at. (2017), Fig. 9

Complete Inception v4 model.



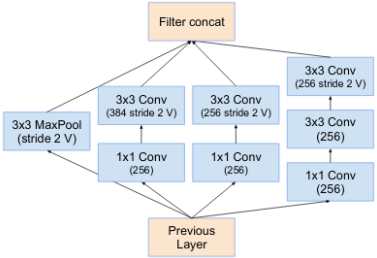
Szegedy et at. (2017), Fig. 10

35×35 Inception ResNet module (Inception-ResNet-A).



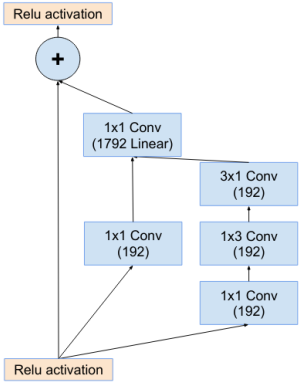
Szegedy et at. (2017), Fig. 11

17×17 Inception ResNet module (Inception-ResNet-B).



Szegedy et at. (2017), Fig. 12

17×17 to 8×8 reduction (Reduction-B) for Inception-ResNet-v1.

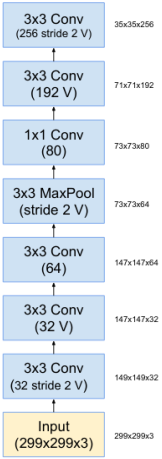


Szegedy et at. (2017), Fig. 13

8×8 Inception-ResNet module (Inception-ResNet-C).

CNNs for image classification

Inception-ResNet (2016)

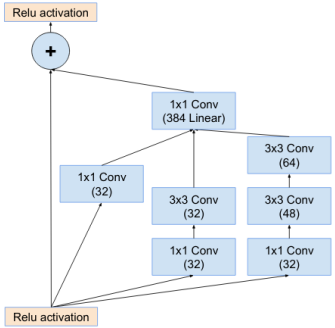


Szegedy et at. (2017), Fig. 14

Stem of Inception-ResNet-v1.

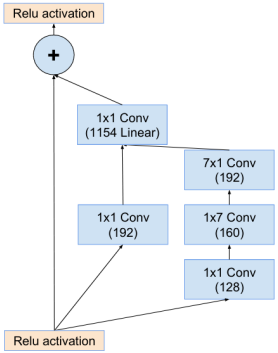
Szegedy et at. (2017), Fig. 15

Complete Incpetion-ResNet-v1 and Inception-ResNet-v2 architectures.



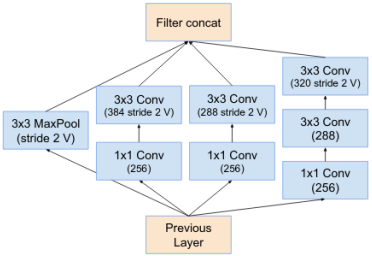
Szegedy et at. (2017), Fig. 16

35×35 module for Inception-ResNet-v2 (Incpetion-ResNet-A).



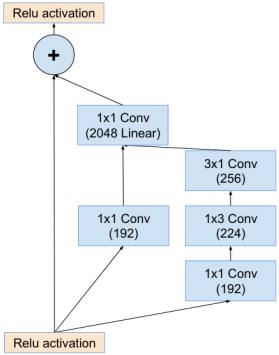
Szegedy et at. (2017), Fig. 17

17×17 module for Inception-ResNet-v2 (Inception-ResNet-B).



Szegedy et at. (2017), Fig. 18

17×17 reduction to 8×8 module for Inception-ResNet-v2 (Incpetion-ResNet-A).



Szegedy et at. (2017), Fig. 19

8×8 module for Inception-ResNet-v2 (Inception-ResNet-C).

Conclusion from Inception-ResNet: training of Inception models is faster with residual connections.

Besides techniques for exploiting residual connections in Inception modules, the paper introduces the idea of residual scaling.

Residual scaling was prompted by the finding that with large numbers of filters, regardless of learning rate, later layers started to produce only zeros due to unstable updates.

Weighting the residuals added to a module's output by 0.1–0.3 before adding improved stability of training.

The classification challenge continued in 2016 and 2017.

In 2016, the Trimps-Soushen team (sponsored by the Chinese Ministry of Public Security) won with 2.99% top-5 error rate.

Trimps-Soushen used a fusion strategy combining the results of various Inception and ResNet models, weighted by their accuracy.

The final recognition challenge was won with 2.25% error rate by a team from Momenta (a Chinese self-driving car company) and Oxford.[4]

The model is called a squeeze-and-excitation network (SENet).

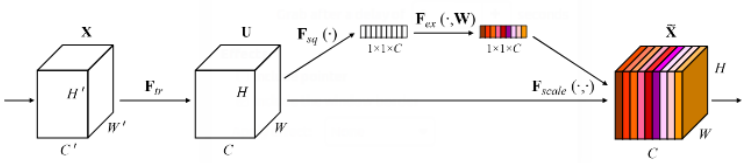
Hu, J., Shen, L., and Sun, G. (2018), Squeeze-and-excitation networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

[4] Trained Caffe models available at `https://github.com/hujie-frank/SENet`.

Basic idea: reweight (scale) channels globally according to informativeness. The technique is called feature recalibration.

We have a transformation $\mathtt{F} : \mathcal{X} \mapsto \mathcal{U}$ with $\mathcal{X} = \mathbb{R}^{H' \times W' \times C'}$ and $\mathcal{U} = \mathbb{R}^{H \times W \times C}$.

A squeeze operation aggregates feature map $\mathtt{U} \in \mathcal{U}$ across dimensions $H \times W$ to produce a channel descriptor vector z. It's a simple average of the elements in each channel (no parameters).



Hu, Shen, and Sun (2018), Fig. 1

Excitation is more complicated...

A squeeze is followed by an excitation operation that performs a sample-specific reweighting of the channels of $\mathtt{U}$.

$$\mathtt{s} = \mathtt{F}_{ex}(\mathtt{z}, \mathtt{W}) = \sigma(g(\mathtt{z}, \mathtt{W})) = \sigma(\mathtt{W}_2\delta(\mathtt{W}_1\mathtt{z}))$$

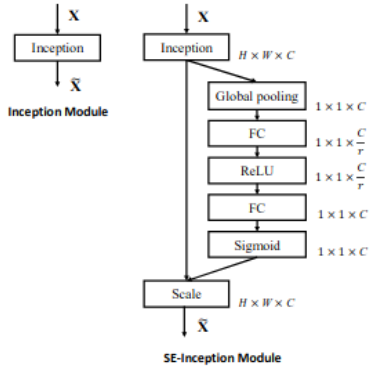
$\delta(\cdot)$ is ReLU, and $\sigma(\cdot)$ is the logistic sigmoid.

$\mathtt{W}_1$ performs dimensionality reduction, and $\mathtt{W}_2$ performs dimensionality expansion.

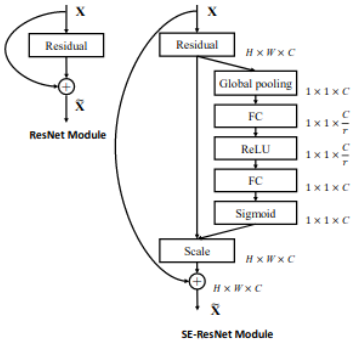
The elements of s are then used to scale $\mathtt{U}$.

For SE to be useful, we can see that $\delta(\mathtt{W}_1\mathtt{z})$ should be a low-dimensional coding of the global activity of each feature map based on which $\mathtt{W}_2$ will decide which feature maps to weight more or less actively.

SE can be applied to simple CNNs like AlexNet easily. The authors also show how to apply SE to Inception modules and ResNet.



Hu, Shen, and Sun (2018), Fig. 2



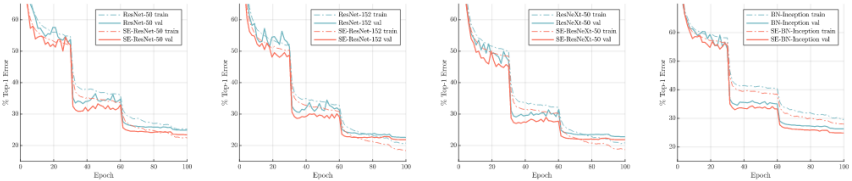
Hu, Shen, and Sun (2018), Fig. 3

The approach improves the single-crop ImageNet performance of several models:

	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [13]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [13]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [13]	23.0	6.7	22.42	6.34	11.30	$21.57_{(0.85)}$	$5.73_{(0.61)}$	11.32
ResNeXt-50 [19]	22.2	-	22.11	5.90	4.24	$21.10_{(1.01)}$	$5.49_{(0.41)}$	4.25
ResNeXt-101 [19]	21.2	5.6	21.18	5.57	7.99	$20.70_{(0.48)}$	$5.01_{(0.56)}$	8.00
VGG-16 [11]	-	-	27.02	8.81	15.47	$25.22_{(1.80)}$	$7.70_{(1.11)}$	15.48
BN-Inception [6]	25.2	7.82	25.38	7.89	2.03	$24.23_{(1.15)}$	$7.14_{(0.75)}$	2.04
Inception-ResNet-v2 [21]	$19.9^{\dagger}$	$4.9^{\dagger}$	20.37	5.21	11.75	$19.80_{(0.57)}$	$4.79_{(0.42)}$	11.76

Hu, Shen, and Sun (2018), Table 2

The performance improvement is immediate and consistent:



Hu, Shen, and Sun (2018), Fig. 4

SE also improves the performance of smaller networks aimed at mobile devices:

	original		re-implementation				SENet			
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	MFLOPs	Params	top-1 err.	top-5 err.	MFLOPs	Params
MobileNet [64]	29.4	-	28.4	9.4	569	4.2M	$25.3_{(3.1)}$	$7.7_{(1.7)}$	572	4.7M
ShuffleNet [65]	32.6	-	32.6	12.5	140	1.8M	$31.0_{(1.6)}$	$11.1_{(1.4)}$	142	2.4M

Hu, Shen, and Sun (2018), Table 3

Lessons to be learned from ILSVRC entries:

- Use small convolutions
- Go deeper
- Reduce dimensionality and number of parameters when possible
- Combine multiple models for improved performance
- Learn residuals rather than direct mappings
- Amplify informative channels

Outline

Next we'll learn about deep learning models for object detection.

Things begin with the R-CNN from UC Berkeley:

- Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014), Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.
- *Selective search* is used to generate region proposals (approx. 2000 per image).
- Each proposed region is scaled to constant size then classified.
- Selective search performs hierarchical clustering of image regions beginning with initial regions from Felzenshwab et al.'s (2004) graph-based pixel clustering technique.

Sample oversegmentation with Felzenszwalb and Huttenlocher (2004)'s method:

Felzenszwalb and Huttenlocher (2004), Fig. 4

Selective search (Uijlings, van de Sande, Gevers, and Smeuers, IJCV, 2013) considers many possible groupings of the small image segments into regions.

The R-CNN applies AlexNet trained on ImageNet to obtain a 4096-element descriptor of the region (the last fully-connected hidden layer).

The 4096-element descriptor is classified with a SVM.

Fast R-CNN and Faster R-CNN improve on the ideas of the R-CNN.

Faster R-CNN (Ren, He, Girshick, and Sun, NIPS, 2015) combines region proposal creation with feature extraction with shared convolutional layers.

Girshick is now at Facebook AI Laboratory (FAIR). Among the recent methods from the group are Mask R-CNN and TensorMask semantic instance segmentation methods.

Before moving to instance segmentation, we look at bounding box regression via YOLO.

Redmon, J., Divvala, S.K., Girshick, R.B., and Farhadi, A. (2016), You only look once: Unified, real-time object detction. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

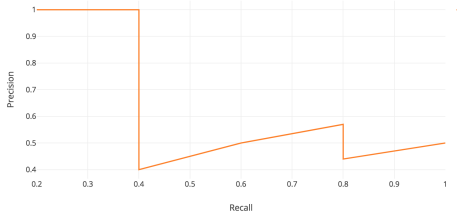
YOLO tries to simplify previous approaches to object detection with a simple structure: a single CNN that predicts bounding boxes for objects, followed my non-maximum suppression:

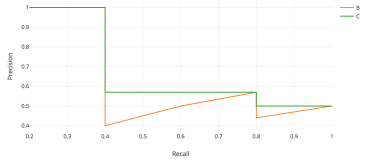
Redmon et al. (2016), Fig. 1

Aside: what is mean average precision (mAP)? First, what is average precision:

- Take the detections of the model, sort by predicted confidence level.
- Classify each detection as correct or incorrect using a measure such as 50% IoU. Only count the most confident overlapping box as correct.
- For each rank n, calculate the precision and recall under the top n detections.
- Plot the curve, smooth by replacing every point by the largest value to the right, and calculate the area under the smoothed curve using interpolation over increments of 0.1 (VOC) or 0.01 (COCO).

Precision recall curve and smoothed precision recall curve:



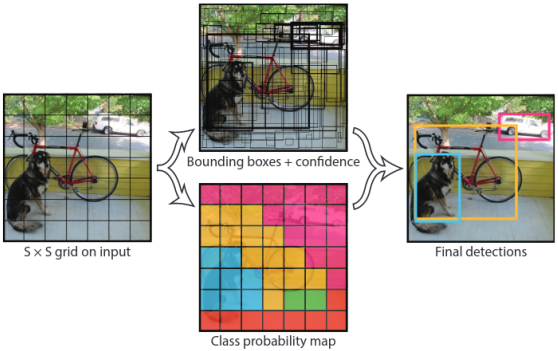


https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

mAP means averaging the AP over multiple IoU thresholds, for example 0.5 to 0.95 by increments of 0.05, and every class.

Model structure: $S \times S$ grid. Each grid cell is responsible for predicting the objects centered in that grid cell.

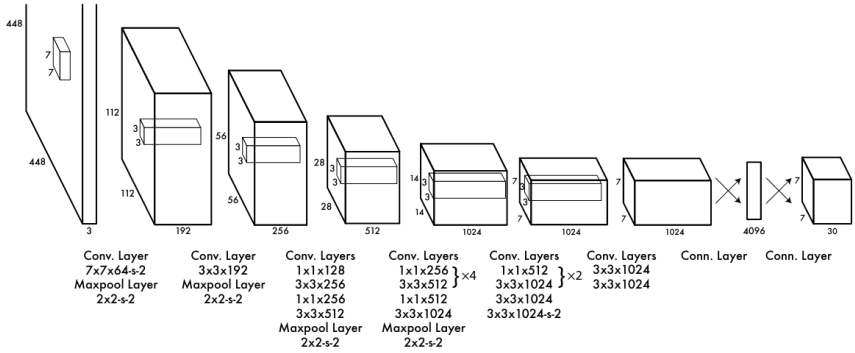
Every grid cell predicts exactly B bounding boxes.



Redmon et al. (2016), Fig. 2

The network reduces resolution from 448×448 to 7×7 with max-pool layers and two stride-2 convolutions.

1×1 reduction convolutions are inspired by Inception/GoogLeNet.



Redmon et al. (2016), Fig. 3

Setup for PASCAL VOC (20 classes):

- Leaky ReLU with 0.1 negative-region slope used everywhere except output.
- 7×7 grid
- $B = 2$ bounding boxes per grid cell
- Bounding box outputs: x, y, w, h scaled to 0–1. x, y are relative to the grid cell; w, h are relative to the image. During training, target is set to bounding box of best matching ground truth object.
- Confidence output scaled to 0–1. Target is set to IoU with best-matching ground truth object.
- A set of class probability outputs is shared by all bounding boxes in the grid (only one class can be predicted per grid cell).
- Output tensor is therefore $7 \times 7 \times 30$ $(5 \cdot 2 + 20)$.

Pretraining:

- First 20 convolutional layers are pretrained for image classification.
- Average pooling then FC layer are added to the convolutional network.
- Input image is downscaled to 224×224.
- Training for approximately one week on ImageNet using DarkNet.
- Single-crop top-5 accuracy of 88%.

Training:

- Input resolution is doubled to 448×448.
- Last four conv layers and two FC layers are added.
- Output layer is linear; cost function is sum squared error.

Some tweaks to the loss function are required to balance emphasis on bounding box accuracy vs. localization error:

- Hyperparameter λ_{coord} is used to increase weight of bounding box values (value of 5 is used)
- Hyperparameter λ_{noobj} is used for confidence output for boxes without ground truth objects (value of 0.5 is used).
- $\sqrt{w}$ and $\sqrt{h}$ are predicted rather than w, h to increase emphasis on small errors for smaller boxes.

Limitations:

- Multiple small objects close together cannot be detected ($B = 2$)
- Unusual aspect ratios or configurations are not recognized well.
- Localization error tends to be high.

Fast YOLO:

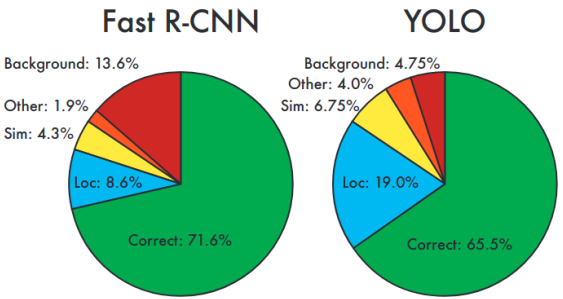
- Similar model with only 9 conv layers.
- Achieves 155 fps on Titan X GPU compared to 45 fps for baseline YOLO.

Comparison of YOLO and Fast YOLO with real time and non-real-time detectors in terms of speed (FPS) and accuracy (mAP):

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Redmon et al. (2016), Table 1

Comparison of Fast R-CNN and YOLO errors on VOC 2007:



Redmon et al. (2016), Fig. 4

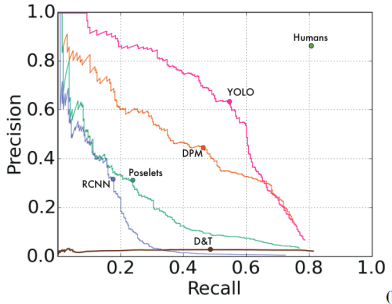
Most YOLO errors are related to localization. False positive rate in background is better.

Public leaderboard on VOC 2012:

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Redmon et al. (2016), Table 3

Generalization to artwork is an indicator of robustness. Models trained on VOC 2007 or 2012 are tested on VOC 2007's "person" test set, Picasso, and People-Art datasets. YOLO performs very well.



(a) Picasso Dataset precision-recall curves.

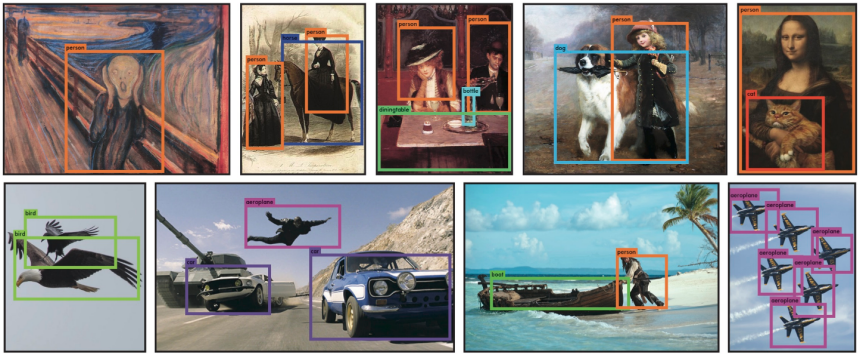
	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

Redmon et al. (2016), Fig. 5

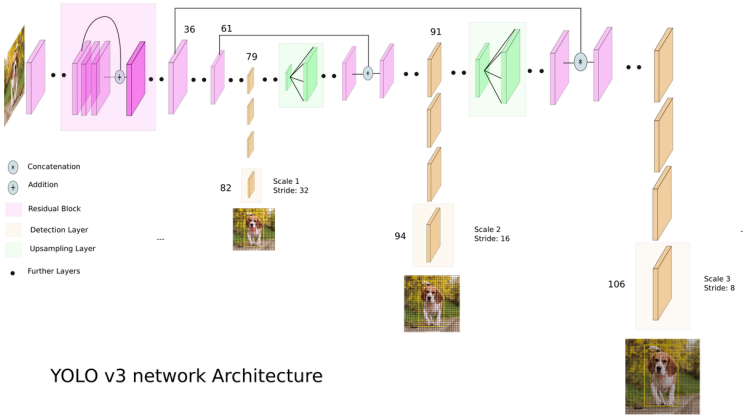
CNNs for object detection

YOLO



Redmon et al. (2016), Fig. 6

YOLO Version 3, 2018: among the fastest and most accurate object detectors available.



YOLO v3 network Architecture

https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

Redmon, J. and Farhadi, A. (2018), YOLOv3: An incremental improvement. *arXiv*, `https://arxiv.org/abs/1804.02767`.

The technical report on YOLO v3 introduces many small changes to turn YOLO v2 into a state-of-the-art detector.

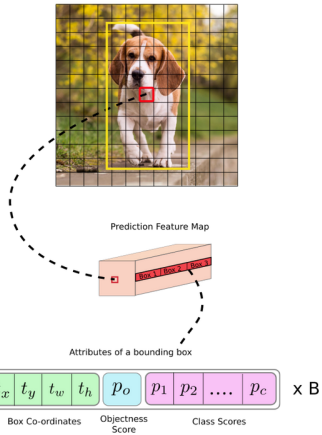
Output is constructed by 1×1 detection kernels applied to feature maps at 3 scales.

Each feature map cell predicts 3 bounding boxes.

Objectness score and bounding box parameter predictors are logistic units rather than linear.

Class predictions are simple logistic units without softmax.

Feature extractors are 3×3 and 1×1 convolutions, with skip connections and residual learning.



https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/

Other tweaks:

- Anchor boxes are obtained by clustering ground truth bounding boxes using k-means. $k = 9$ clusters were separated into 3 scales. Each cell has one bounding box predictor for each of the 3 anchor boxes. Predictions are relative to the anchor box's parameters.
- Multiscale processing begins with the convolutional feature map at the coarsest scale. It is upscaled then concatenated with with a previous feature map of the correct size.
- Later scales are finer, performed by another upscaling and concatenatation with earlier feature maps of the correct size.
- Feature extractor is Darknet-53, which is better and slower than Darknet-19 but inferior to and faster than ResNet 101 or ResNet 152.

Final performance is comparable to SSD (better at IOU 50) with 3x speed.

Performance at IoU 50 vs. other levels of localization accuracy indicates that YOLO v3 still has issues with localization despite improvements.

Besides the R-CNN family and YOLO:

- SSD: Single shot multibox detector, from several universities plus Google, uses VGG16 for the feature detector, then 4 object predictions for each cell in the feature map.
- S^3FD: Single-shot scale invariant face detector, from CASIA, similar principles with multiscale anchors.

Outline

There are many methods for segmentation in computer vision. For example:

- Active contours begin by initializing an arbitrary object contour then iteratively moving the contour to minimize an energy functional.
- Energy-based methods iteratively minimize an energy function encouraging similar neighboring pixels to be grouped together and dissimilar neighboring pixels to be split into separate segments.

These techniques could benefit from learning in order to determine the free parameters of the enery models, but they do not require learning.

Learning is more central, however, to methods that segment images based on local properties such as color, intensity, texture, and composition of parts:

- In pixel-based segmentation, we build a model offline then use the model to segment images into regions based purely on pixel properties.
- Supervised mask-based methods typically use deep learning methods to obtain detailed per-pixel categories (semantic segmentation) and identity (instance-aware semantic segmentation).

Pixel based segmentation methods may be supervised or unsupervised.

Unsupervised methods:

- Perform clustering of pixels based on frequency and color information.
- Simplest example: Otsu's binarization method applied to multiple images.
- Other methods include k-means, fuzzy c-means, Gaussian mixture models.

Supervised methods:

- Use a labeled training set to obtain a generative probabilistic model or direct decision boundary induction technique.
- Example: the H-S histogram based skin pixel classifier we saw earlier in these lecture notes.

We will see the skin pixel classifier at work in a tutorial.

The utility of pixel-based methods is very limited, however. Next we see how learning can be exploited in region-based segmentation.

Outline

Region-based segmentation methods use local neighborhood information and priors to improve segmentation compared to simplistic pixel-based methods.

Most useful segmentation methods are region based.

Example: unsupervised learning with mean shift:

- Performs clustering of vectors representing image pixels by their color and spatial coordinates.
- Image pixels are treated as samples from a distribution over this space.
- A non-parametric model of the joint space-color distribution of pixels in this space is formulated from the sample.
- Gradient ascent is used to find modes of the distribution.

Mean shift is a simple algorithm to find the mode of a distribution represented by a sample and an initial estimate of the mode:

1. Begin with an initial estimate x.
2. Construct a kernel function $K(\cdot)$, for example
$$K(\mathrm{x}_i - \mathrm{x}) = \exp(-c\|\mathrm{x}_i - \mathrm{x}\|^2).$$
3. Given the neighborhood $N(\mathrm{x})$ where $K(\cdot)$ is nonzero, compute
$$\mathrm{m}(\mathrm{x}) = \frac{\sum_{\mathrm{x}_i \in N(\mathrm{x})} K(\mathrm{x}_i - \mathrm{x})\mathrm{x}_i}{\sum_{\mathrm{x}_i \in N(\mathrm{x})} K(\mathrm{x}_i - \mathrm{x})}.$$
4. Let $\mathrm{x} \leftarrow \mathrm{m}(\mathrm{x})$ and repeat from 3 until convergence.

[Sample applications: clustering regions in an image by color, tracking an object in a video sequence.]

As before, learning can be useful in determining appropriate parameters for methods such as mean shift.

However, if color and position alone are insufficient, learning may be more important.

We may require more sophisticated neighborhood texture models and means to classify those neighborhoods.

Example neighborhood texture model: SIFT descriptors.

Learning problem: which descriptors are characteristic of regions of interest and which are not?

Region-based segmentation

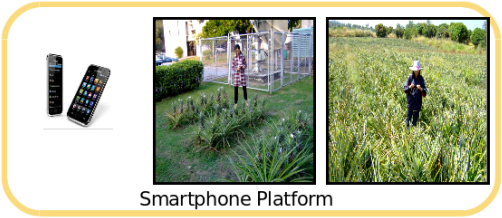
Application to crop mapping

Example from AIT: apply 3D vision (structure from motion) and texture-based region segmentation for fruit in the field for purposes of yield prediction.

Image sequence extracted from video file taken from agricultural field

Region-based segmentation

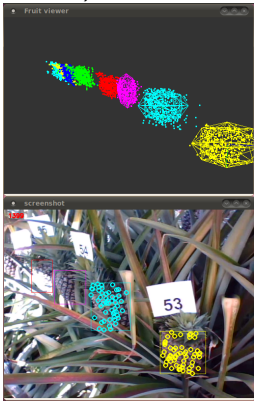
Application to crop mapping

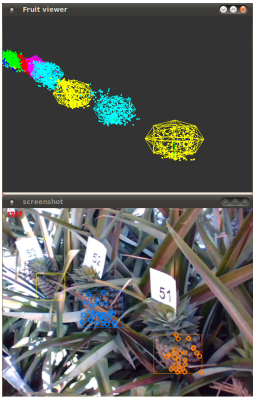


Region-based segmentation

Application to crop mapping

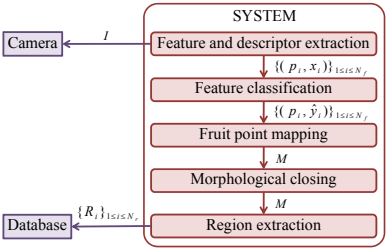
We apply keyframe selection to the video sequence, find fruit regions (using keypoint descriptor classification), and perform 3D reconstruction of the point regions.





Region-based segmentation

Application to crop mapping



Fruit detection system

Extracting feature points from agricultural image.

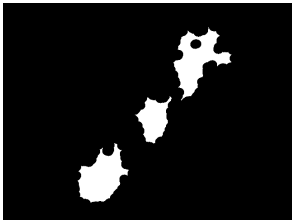
Algorithm:

1. Acquire input image I.
2. Apply interest point operator to obtain N_f candidate feature points $\{\mathrm{p}_i\}_{1 \leq i \leq N_f}$.
3. For each feature point p_i, obtain a feature descriptor vector x_i.
4. For each feature descriptor vector x_i, obtain predicted label $\hat{y}_i$ (1 for positive or 0 for negative).
5. Create binary image M the same size as I and set all pixels to 0.
6. For each p_i for which $\hat{y}_i = 1$, set $M(\mathrm{p}_i)$ to 1.
7. Perform morphological closing with appropriately shaped structuring element on M.
8. Extract connected components from M and return large positive regions $\{R_i\}_{1 \leq i \leq N_r}$, as output.

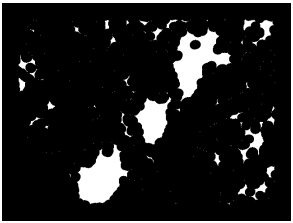
Region-based segmentation

Application to crop mapping

Predicted positive feature points



Connected components results



Filtering results

Detected regions

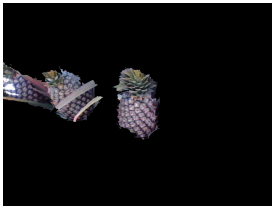
Another possible platform is a UAV.

Here the goal is to obtain high quality 3D maps of a crop from a low resolution monocular camera mounted on a UAV.

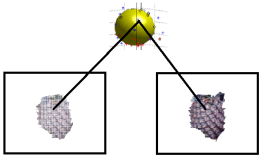
Quad-copter

Image taken from quad-copter

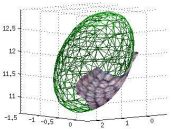
Here we combine dense region-based segmentation with a SIFT descriptor classifier and model-based 3D reconstruction of spheroids (restricted quadrics).



Dense segmentation with SIFT classifier

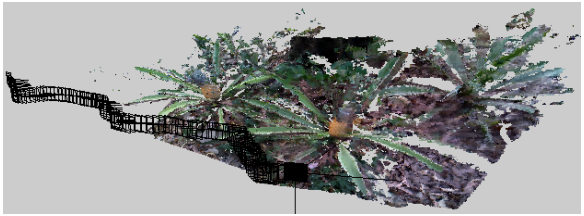


Optimization of spheroid to minimize SIFT distance



Optimized spheroid

We are working on Fast Fusion modeling of agricultural crops with RGBD sensors:



Outline

CNNs for segmentation

Semantic segmentation

CNN models have emerged as extremely effective for region-based segmentation:

- Semantic segmentation methods use fully convolutional networks (FCNs) to classify every pixel in a scene.
- Instance segmentation methods combine an object detection pipeline with semantic segmentation inside the detection region.

Typical semantic segmentation methods:

- Object classification CNN backbone (e.g., ResNet-101) as an encoder. Fully connected layers are removed after training on a classification dataset.
- Optionally, the encoder's output is progressively upscaled with a fully convolutional decoder for spatial accuracy.
- Final layer gives each pixel/grid cell a meaningful object category.

Sample state of the art method: U-Net (MICCAI 2015).

Instance-aware segmentation additionally aims to give unique labels to different images in the scene.

One of the most effective methods is Mask R-CNN, which combines Faster R-CNN with an instance mask for each region proposal.

He, K., Gkioxari, G., Dollar, P., and Girshick, R. (2017), Mask R-CNN. *IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969.

See also: Chen, X., Girshick, R., He, K., and Dollar, P. (2019), TensorMask: A Foundation for Dense Object Segmentation

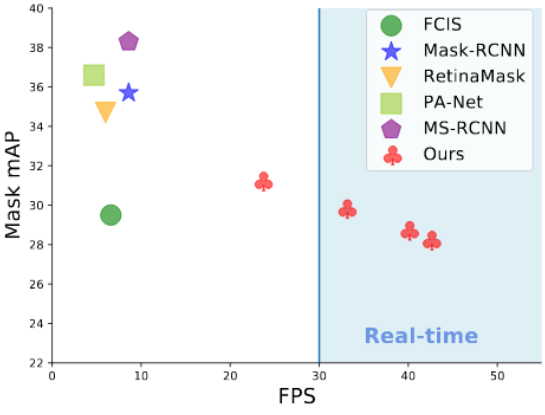
Until recently, all deep learning instance segmentation methods were too slow for real time applications.

However, the first real time instance segmentation CNN, YOLACT, streamlines the Mask R-CNN idea similarly to how YOLO streamlines Faster R-CNN.

Bolya, D., Zhou, D., Xiao, F., and Lee, Y.J. (2019), YOLACT: Real-time Instance Segmentation, *IEEE International Conference on Computer Vision (ICCV)*

CNNs for segmentation

YOLACT



Bolya, Zhou, Xiao, and Lee (2019), Fig. 1

Bolya et al. (2019) introduce YOLACT, the fastest-ever instance segmentation method.

29.8 mAP on the Microsoft COCO instance segmentation dataset (about 20% worse than Mask R-CNN).

33.5 fps on a Titan Xp GPU (300% faster than Mask R-CNN)!

Why wasn't instance segmentation real time until now?

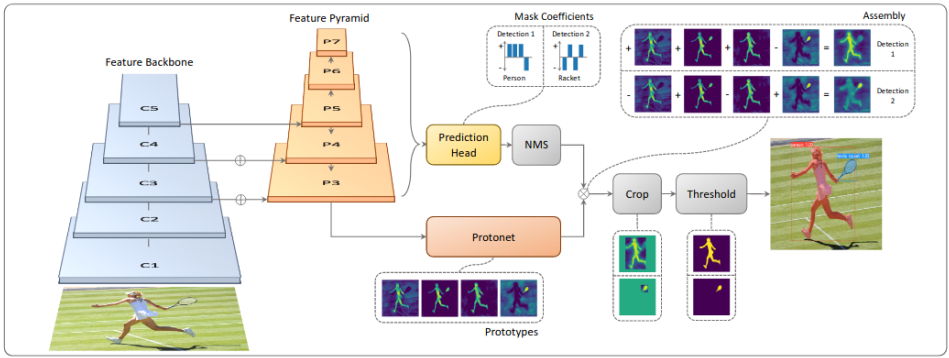
Mask R-CNN is a two-stage method:

- Generate regions of interest (RoIs) in the first stage
- Classify and segment each RoI in the second stage after a transforming with a ROI repooling step (sequential, slow).

Bolya et al.'s idea:

- Skip localization.
- Generate a dictionary of non-local prototype masks over the entire image.
- Predict a set of linear combination coefficients per instance.
- Linearly combine the prototypes using predicted coefficients.
- Crop with a predicted bounding box.

YOLACT uses two parallel streams: FCN protonet for full-image prototype masks and a prediction head with per-instance prediction of mask coefficients (in addition to the bounding box and class predictions).



Bolya, Zhou, Xiao, and Lee (2019), Fig. 2

Why is this faster than Mask R-CNN?

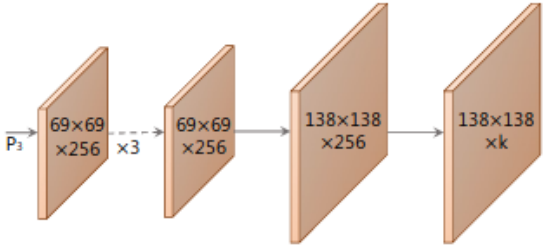
Mask R-CNN computes feature maps then region proposals.

The masking part of the model has to wait for region proposal process to complete.

By generating masks over the entire image, masking does not wait for region proposals (RPs are calculated in the prediction head using fully connected units).

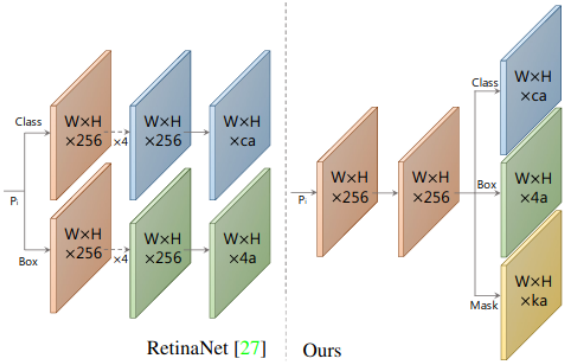
The protonet is like the output layer of a semantic segmentation FCN.

The protonet's features are taken from a Feature Pyramid network then upsampled by a factor of 2×2:



Bolya, Zhou, Xiao, and Lee (2019), Fig. 3

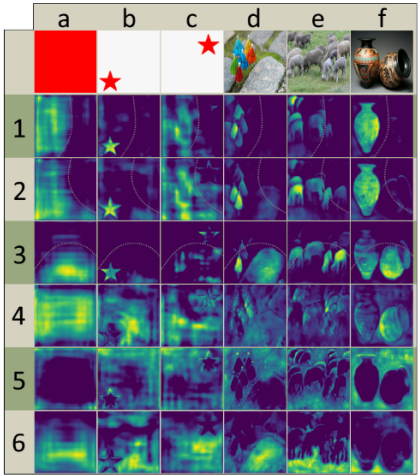
In addition to the two typical branches (class identity and bounding box parameters), the prediction head contains a third branch to predict k mask coefficients (one for each mask in the protonet).



Bolya, Zhou, Xiao, and Lee (2019), Fig. 4

CNNs for segmentation

YOLACT



Bolya, Zhou, Xiao, and Lee (2019), Fig. 5

Sample prototypes that emerge from training in the protonet stream.

CNNs for segmentation

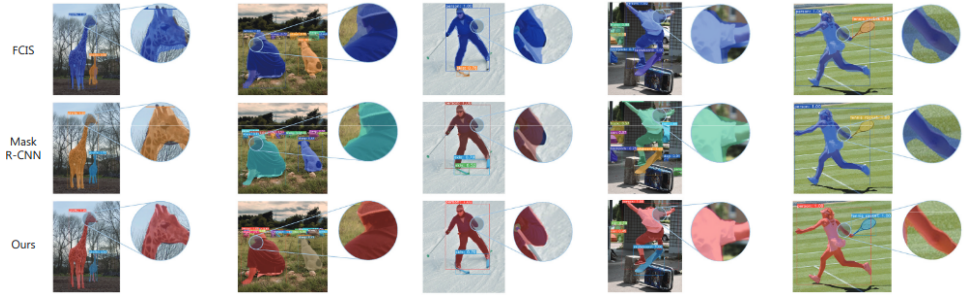
YOLACT

Bolya, Zhou, Xiao, and Lee (2019), Fig. 6

Several other improvements are combined: (fast non-maximum suppression based on parallel computation of IoU on the GPU, extra semantic segmentation training losses not used at runtime).

Results are awesome...

Mask quality is very high:



Bolya, Zhou, Xiao, and Lee (2019), Fig. 7

More results: