Computer Vision
Designing, Visualizing and Understanding Deep Neural Networks

CS 182/282A

Guest Lecture: Kumar Krishna Agrawal
10/04/2022

Instructor: Anant Sahai
UC Berkeley
CNNs : Training, Debugging, Understanding
Review: Skip Connections & ResNets

Training plain-CNN without skip-connections, deeper networks doesn’t optimize to better error%.

CNNs w/ skip connections improves trainability of deeper models, by improving gradient flow.


Pseudocode

```
def resnetBlockV1(x):
    op = nn.Sequential(
        Conv2D,
        BatchNorm2D,
        ReLU,
        Conv2D,
        BatchNorm2D)
    out = x + op(x)
    return ReLU(out)
```

\[
\frac{dH}{dx} = \frac{dF}{dx} + I
\]

regulate ill-conditioned gradients
Training : How fast?

Natural followup question:

Well, what are challenges in speeding up model training?

→ optimization challenges:
  × use same learning rate schedule
  ✓ linearly scale LR
→ increasing batch size:
  × activations don’t fit memory
  ✓ distributed dataparallel training
→ data augmentation:
  × overfitting to train-set fast, does not necessarily improve generalization
  ✓ regularize with data-augmentations

Training Resnet-50 models on Imagenet [1.2M samples] with 8 A100 GPUs, typically requires around <20min to achieve > 75.2% test-set accuracy. More details at https://github.com/mosaicml/benchmarks/

Goyal et al. “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour” 2017
ResNet50 recipes: https://github.com/mosaicml/benchmarks/
For a given network architecture (ResNet), the error approximately scales as a power law with FLOPs (linear fit on the log-log curve). Scaling includes width-multipliers, depth and image resolutions.

What do CNNs learn? An instance-level view

**Intuition**: “Easy” samples should be learned faster

How to characterize example-difficulty?
Consider dataset \( \mathcal{D} \sim_n P \) where we take \( n \) iid-samples, define the consistency-profile for an instance \((x, y)\) as

\[
C_{P,n}(x, y) = \mathbb{E}_{\mathcal{D} \sim_n P} \left[ \mathbb{P} \left( f(x; \mathcal{D}\setminus\{(x, y)\}) = y \right) \right]
\]

Regularities and exceptions in a binary chairs vs non-chairs problem

Consistency profiles across samples.

On the ImageNet dataset, instances are ordered by estimates of C-score, from regularities (high C-score) to exceptions (low C-score).

Jiang et al. “Characterizing Structural Regularities of Labeled Data in Overparameterized Models” 2020
Example Difficulty & Depth-Complexity

For representations at intermediate layer, we look at the predictions of k-NN classifier.

\[
prediction_{\text{depth}}(x, f_\theta) = \arg\min_{l \in L} f(x, \theta_l) = f(x, \theta_{>l})
\]

### Some takeaways

1. Prediction depth is a good proxy for example difficulty.
2. “Easier” examples are learned “first”, both in terms of optimization (earlier epochs) and model (earlier layers).
3. Estimating example difficulty allows us to early-exit, saving compute at inference!

Beyond Image Classification
convolutional networks: map image to output value

So far ...

e.g., semantic category (“bicycle”)
Images are rich-source of information about the environment.

In practice:
- identify all distinct objects in a scene (multi-output)
- find object position beyond labels
- from 2D to 3D (e.g. depth estimation from images)

https://github.com/facebookresearch/detectron2
Standard computer vision tasks

- object classification
- object localization
- object detection
- semantic segmentation
Object Localization
The Problem Setup

Previously: $\mathcal{D} = \{x_i, y_i\}$

- image
- label $l_i$

Now: $\mathcal{D} = \{x_i, y_i\}$

- image
- $(l_i, x_i, y_i, w_i, h_i)$

**convolutional networks:** map **image** to **output value**

- e.g., label, object location, BoundingBox
Measuring localization accuracy

Did we get it right?

**Intersection over Union (IoU)**
Different datasets have different protocols, but one reasonable one is: correct if \( \text{IoU} > 0.5 \)

If also outputting class label (usually the case): correct if \( \text{IoU} > 0.5 \) and class is correct

**Note**: Not the loss function! Metric for evaluation model quality

\[
\text{IoU} = \frac{\text{I}}{\text{U}}
\]
Object Localization as Regression

\[ D = \{(x_i, y_i)\} \quad y_i = (\ell_i, x_i, y_i, w_i, h_i) \]

→ simple design
→ train either jointly (multi-task, more on this later), or use pretrained classification model, then train regression head
→ Not usually the standard approach, we’re not using structure of the problem.
What if we classify every patch in the image?
Sliding Windows

\[ D = \{(x_i, y_i)\} \quad y_i = (\ell_i, x_i, y_i, w_i, h_i) \]

could just take the box with the highest class probability

more generally: non-maximal suppression
A practical approach: OverFeat

\[ \mathcal{D} = \{(x_i, y_i)\} \quad y_i = (l_i, x_i, y_i, w_i, h_i) \]

- **Pretrain on just classification**
- **Train regression head on top of classification features**
- **Pass over different regions at different scales**
- **“Average” together the boxes to get a single answer**

Sliding windows & reusing calculations

Sliding window **classification** outputs at each scale/position (*yellow* = bear)

Predicted box x, y, w, h at each scale/position (*yellow* = bear)

Final combined bounding box prediction (*yellow* = bear)

Sermanet et al. “**OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks.**” 2013
Problem: sliding window is very expensive! (36 windows = 36x the compute cost)

This looks a lot like convolution...

Can we just reuse calculations across windows?

“Convolutional classification”

before: 2x fully connected layers with 4096 units

now: 2x **convolutional** layers with 1 x 1 x 4096 filters
Sliding windows and reusing computation

This kind of calculation reuse is extremely powerful for localization problems with conv nets.

We'll see variants of this idea in every method we'll cover today!
Summary

**Building Block**: Conv Net that outputs class and bounding box coordinates

**Evaluate** this network at multiple scales and for many different crops, each one producing a probability and bounding box

**Implement** the sliding window as just another convolution, with 1x1 convolutions for the classifier/regressor at the end to save on computation
Object Detection
The Problem Setup

Before

Now

\[(x_i, c_{i,1}, x_{i,1}, y_{i,1}, w_{i,1}, h_{i,1}, c_{i,2}, x_{i,2}, y_{i,2}, w_{i,2}, h_{i,2}, \ldots, c_{i,n_i}, x_{i,n_i}, y_{i,n_i}, w_{i,n_i}, h_{i,n_i})\]

number of objects \(n_i\) different for each image \(x_i\)!

rightarrow

“cat” : 0.21

\((x, y, w, h)\)  ???
Dense-Prediction: Generating multiple outputs

**Sliding window:** each window can be a different object

Instead of selecting the window with the highest probability (or merging windows), just output an object in each window above some threshold.

**Big problem:** a high-scoring window probably has other high-scoring windows nearby.

**Non-maximal suppression:** (informally) kill off any detections that have other higher-scoring detections of the same class nearby.

**Actually output multiple things:** output is a list of bounding boxes.

**Obvious problem:** need to pick number, usually pretty small.

- Works great if combined
- Not good by itself

\[
\begin{align*}
\text{non-maximal} \quad \text{maxim} \\
\text{works great if combined} \quad \text{not good by itself}
\end{align*}
\]
Case Study: You Only Live Once [YOLO] Look

Actually, you look a few times (49 times to be exact…)

different output for each of the 7x7 (49) grid cells (a bit like sliding window)

for each cell, output:

1. \((x, y, w, h)\) (confidence)
2. IoU (class label)

zero if no object

output \(B\) of these

some training details:

need to assign which output is “responsible” for each true object during training

just use the “best-fit” object in that cell (i.e., the one with highest IoU)

What if we have too many objects?

Well, nothing… we just miss them

CNNs + Region proposals

This is really slow
But we already know how to fix this!

Girshick et al. "Fast R-CNN." 2015
CNNs + Region proposals

A smarter “sliding window”: region of interest proposals

Compare this to evaluating every location:
CNNs + Region proposals

How to train region of interest proposals?

Very similar design to what we saw before (e.g., OverFeat, YOLO), but now for predicting if any object is present around that location.

Ren et al. “Faster R-CNN.” 2015
Building Efficient Detectors: Feature Pyramids

Key Idea: Aggregate information across multiple scales


design principles
+ multi-scale features
+ skip-connections
+ single-forward pass
new operator
+ up-sampling convolution
Figure 2: **Feature network design** – (a) FPN [23] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P3 - P7); (b) PANet [26] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [10] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) is our BiFPN with better accuracy and efficiency trade-offs.
Suggested readings

  ▪ Just regress to different bounding boxes in each cell
  ▪ A few follow-ups (e.g., YOLO v5) that work better

  ▪ Uses region of interest proposals instead of sliding window/convolution

➢ Ren et al. “Faster R-CNN.” 2015
  ▪ Same as above with a few improvements, like region of interest proposal learning

➢ Liu et al. SSD: Single Shot MultiBox Detector. 2015
  ▪ Directly “classifies” locations with class and bounding box shape

➢ Lin et al. Feature Pyramid Networks for Object Detection, 2016
  ▪ Proposes Feature Pyramid Networks (FPN), aggregating information at multiple scales

  ▪ Compute efficient detectors with Bidirectional FPN (BiFPN)
Semantic Segmentation
The Problem Setup

Problem:
K-class classification problem, predicting a label per pixel $y_i \in \mathcal{S} = \{c_1, c_2, \ldots, c_K\}$

Learning a mapping $f_\theta : \mathbb{R}^{3 \times H \times W} \mapsto \mathcal{S}^{H \times W}$

Note that the output should have the same resolution as the input!

Brute-Force baseline
→ Never downsample (i.e. zero padding, stride 1, no pooling)
→ Computational cost?

Detect all objects in an image

Label every single pixel with its class

Actually simpler in some sense:
• No longer variable #outputs
• Every pixel has a label

Learn “per-pixel” classifier
The Problem Setup

**Task:** Classify every point with a class

Don’t worry for now about instances (e.g., we don’t distinguish between two adjacent cows, they “look” like a “cow blob”. This is okay for now, instance-segmentation deals with different instances)

**The challenge:** Design a network architecture that makes this “per-pixel classification” problem computationally tractable.
Fully Convolutional Networks

**Desiderata:**
set of operations that preserve resolution at output

**Constraints:**
effective *receptive field* of convolution filters grows with depth (early layers have local view)

---

Slide borrowed from Fei-Fei Li, Justin Johnson, Serena Yeung
Conv. Operators: Down/UpSampling

Normal Convolutions: reduce resolution with stride, padding

Dilated Convolutions: Increase receptive-field more rapidly

Transpose Convolutions: increase resolution with fractional "stride"

Un-pooling

Corresponding pairs of downsampling & upsampling layers
Bottleneck architecture

U-Net Architecture

**Problem:**
Downsampling loses information, that upsampling cannot recover.

**Intuition:**
Explicitly append filters from earlier downsampling layer that preserve high-frequency details (such as edges)

Instance Segmentation: Mask R-CNN

Problem:
Can we use RPNs and RCNNs for instance segmentation?

Intuition:
Alongside the class, bounding box predict a mask per region-proposal.

General principle
Masking (learned or imposed) is very useful way of generating irregular predictions.

Effectively Mask R-CNN adds another prediction branch to predict instance-specific masks per RoI.
Standard computer vision tasks

- Object classification
- Object localization
- Object detection
- Semantic segmentation