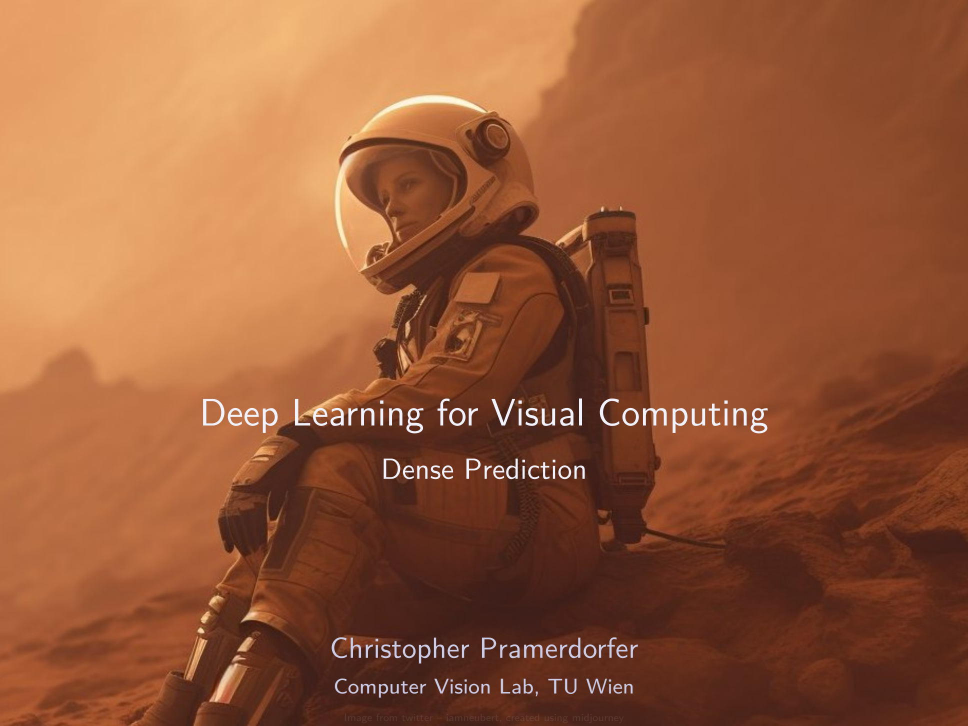

Deep Learning for Visual Computing

Dense Prediction

Christopher Pramerdorfer
Computer Vision Lab, TU Wien

Image from twitter – iamneubert, created using midjourney

Topics

Dense prediction

- Semantic image segmentation
- Keypoint detection
- Image restoration

Building blocks

- Upsampling layers
- U-Nets & Conditional GANs

This Week in AI
Drag Your GAN

Image from youtube

Dense Prediction

We often want to predict something for every input image pixel

- This is called dense prediction
- How can we achieve that?

Obvious solution is to avoid pooling

- We already established this is too slow
- Also receptive field increases slowly so missing context

Dense Prediction

Solution is to use an encoder-decoder architecture

- Encoder learns high-level features
- Decoder maps these features back to original size

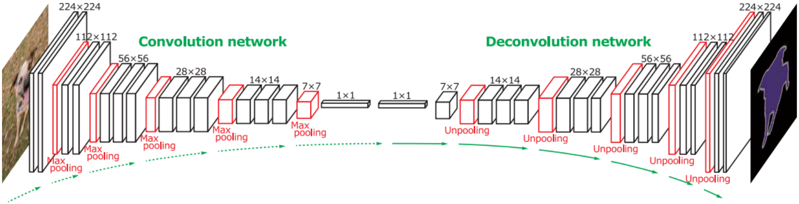

Image from [1]

Dense Prediction

Layer types

- Encoder: conv and pooling layers (e.g. ResNet)
- Decoder: conv and upsampling layers

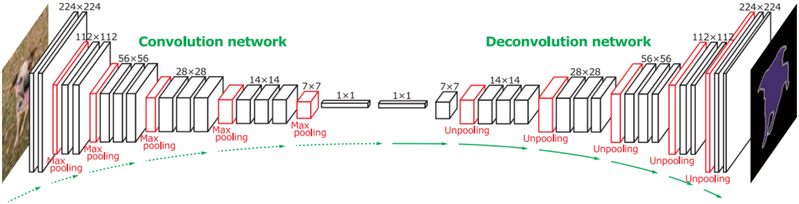

Image from [1]

Dense Prediction
Upsampling – Interpolatuion

One way is to utilize standard 2D upsampling techniques

- Nearest neighbor or bilinear interpolation
- Independently per channel

Fast and simple

Usually paired with a conv layer

- To learn useful features / transformations

Dense Prediction
Upsampling – Interpolation

Example ("inverse max-pooling")

- Upsampling by factor of 2
- Nearest neighbor interpolation

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \implies \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix}$$

Transposed convolutions are convolutions with stride $1/s$

- Also called fractionally strided convolutions
- See link below for animations of variants

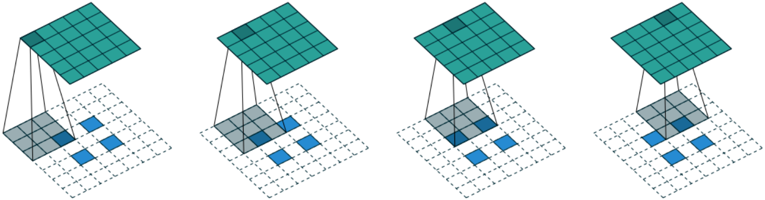

Image from github

Learnable upsampling

- More powerful than pure interpolation
- But often leards to artifacts due to overlapping kernel

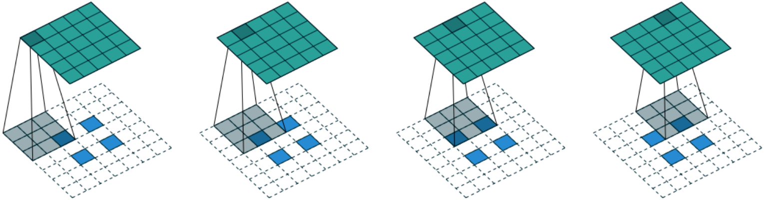

Image from github

Subpixel convolutions are most recent upsampling method

- Usually outperforms other methods covered
- Also called pixel shuffle

Operation

- Given input of size $(C \cdot r^2) \times H \times W$
- Rearange (shuffle) to $C \times rH \times rW$
- Then do a stride 1 convolution

Dense Prediction
Upsampling – Subpixel Convolutions

Example with $r = 3$ and $9 \times 7 \times 7$ input

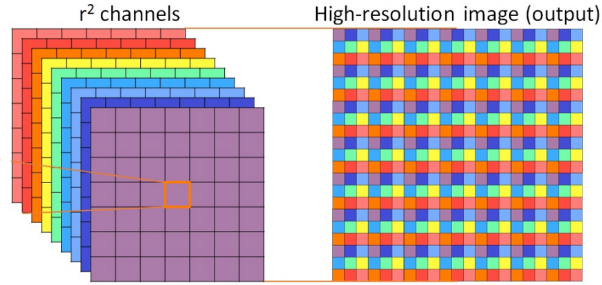

Image from [4]

Dense Prediction

We can train such networks like we did before

- Gradient descent and backpropagation
- Just define a suitable loss depending on task

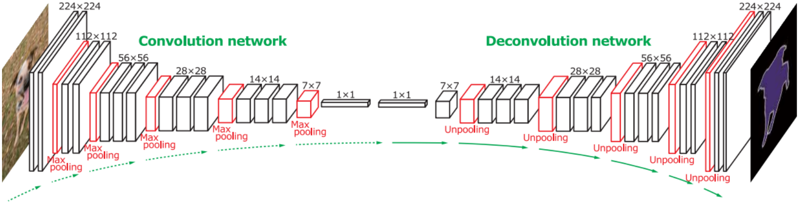

Image from [1]

Dense Prediction

However this would probably not work very well because

- The network has no idea how objects look like
- Spatial information is lost due to excessive downsampling

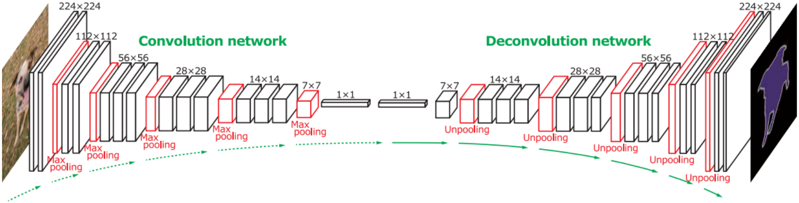

Image from [1]

Dense Prediction

To address this we

- Pre-train the encoder (e.g. ImageNet classification)
- Use skip-connections across the two parts

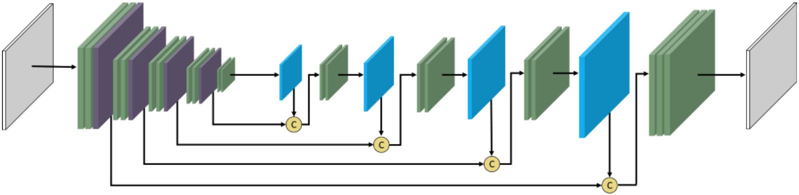

Image from medium.com

A popular method for defining these connections is U-Net [2]

- Skip from before every downsampling layer
- To after every corresponding upsampling layer
- Concatenate (not sum) signals

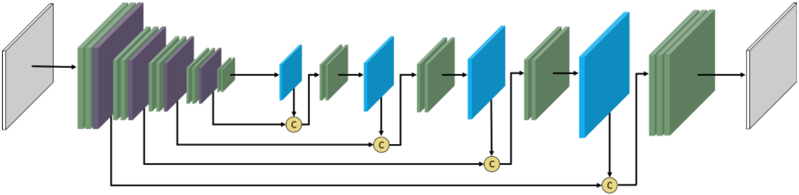

Image from medium.com

Dense Prediction
U-Nets

Skip-connections forward high-resolution data

- Makes spatial information available to decoder
- Signals are concatenated due to different nature

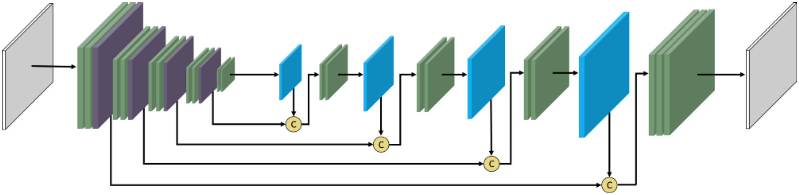

Image from medium.com

Dense Prediction
U-Nets

Modern U-Nets often use

- A ResNet for downsampling
- Subpixel convolutions for upsampling

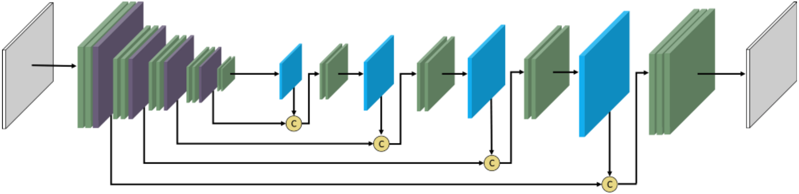

Image from medium.com

Dense Prediction
Applications

We now have all we need to solve various tasks

- Simply choose C of output accordingly
- And use a suitable loss function
- Always pre-train the encoder

We will cover three popular applications

Semantic Image Segmentation

Task definition

- Given T classes
- Assign a class label to each image pixel

Image from cocodataset.org

Semantic Image Segmentation

Implementation

- Configure last layer to output T channels
- Use cross-entropy loss $L_p(\boldsymbol{\theta})$ for every pixel p
- Minimize $L(\boldsymbol{\theta}) = \sum_p L_p(\boldsymbol{\theta})$

Keypoint Detection

Keypoint detection task definition

- Given K keypoints
- Locate each keypoint in image (multiple times)

A popular application is (sparse) human pose estimation

- Keypoints for hands, feet, shoulders etc.

Keypoint Detection

Implementation

- Configure last layer to output K channels
- Each channel encodes keypoint confidences (confidence map)
- Use e.g. L2 loss $L_c(\boldsymbol{\theta})$ (MSE loss) for every channel c
- Minimize $L(\boldsymbol{\theta}) = \sum_c L_c(\boldsymbol{\theta})$

Supporting multiple instances requires keypoint matching

- Can be done in post-processing step

Keypoint Detection

Note that dense prediction is not mandatory

- Could simply regress keypoint coordinates

But dense prediction is more flexible & powerful

- Confidences facilitate downstream tasks
- Can predict additional dense data (e.g. part affinity fields)

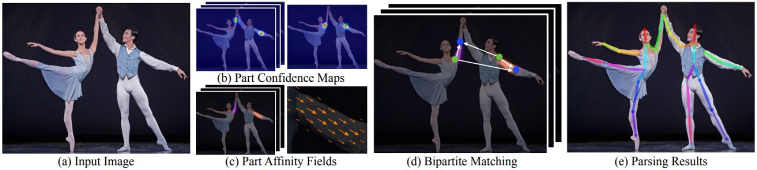

Image from [3]

Keypoint Detection

Human pose estimation with OpenPose [3]

link

Image Restoration

Image restoration aims to improve image quality

Common tasks

- Remove overlays and compression artifacts
- Sharpen or upscale

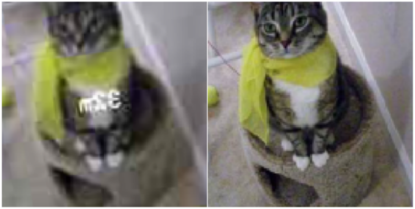

Image from medium.com

Image Restoration

Implementation

- Configure last layer to output C_0 channels (usually $C_0 = 3$)
- Train on pairs of low-quality and high-quality images
- Use L2 loss $L_p(\boldsymbol{\theta})$ for every pixel p
- Minimize $L(\boldsymbol{\theta}) = \sum_p L_p(\boldsymbol{\theta})$

Image Restoration

Generating large datasets is comparatively easy

- Take high-quality images
- Degrade them automatically depending on task (e.g. blur)

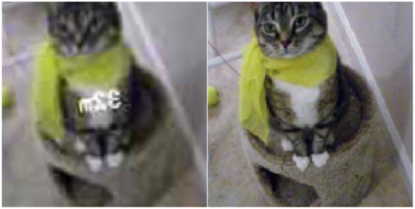

Image from medium.com

Image Restoration

The resulting network would probably be

- Good at removing text and compression artifacts
- Not so good at sharpening the image

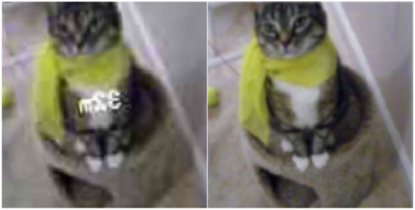

Image from medium.com

Image Restoration

Because L2 loss does not capture sharpness well

- Blurring has little effect on most pixel values
- Per-pixel losses cannot capture overall sharpness

Image from [4]

Image Restoration

Unclear how to define a good loss for sharpness

- Maybe we can "learn" it?

To do so we use an adversarial loss

- Loss function is a binary classifier $\mathcal{D}$ (discriminator or critic)
- That answers "is the given image real or fake?"

In our context of image restoration

- "real" means an original high-quality image
- "fake" means one processed by the network

Image Restoration
GANs

This is a Conditional Generative Adversarial Network (GAN)

- Generator $\mathcal{G}$ (our encoder-decoder) learns to fool the critic
- Critic learns not to be fooled

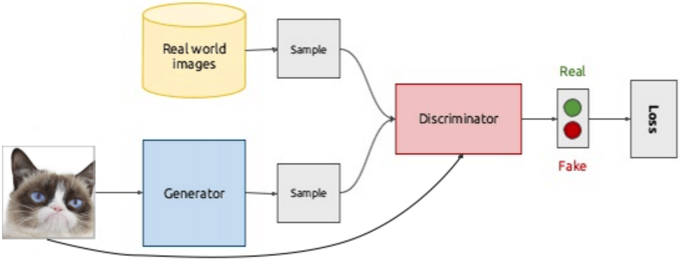

Image from sigmoidal.io

If $\mathcal{G}$ can reliably fool $\mathcal{D}$

- Processed images look just like high-quality ones
- And thus must be high-quality themselves

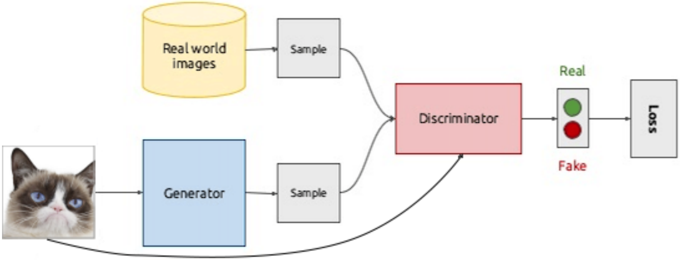

Image from sigmoidal.io

Image Restoration
GANs

$\mathcal{G}$ and $\mathcal{D}$ form a single network during training

- Gradients flow from $\mathcal{D}$ to $\mathcal{G}$
- So $\mathcal{G}$ learns via (cross-entropy) loss on $\mathcal{D}$

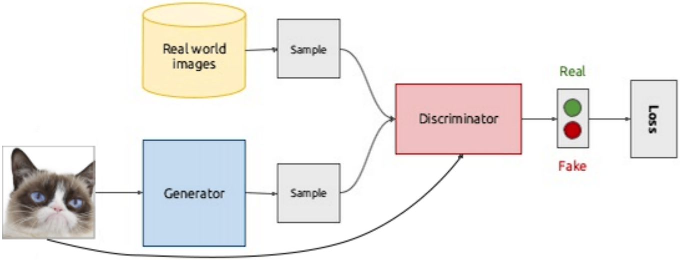

Image from sigmoidal.io

Training is an iterative process

- Train $\mathcal{D}$ (but not $\mathcal{G}$) on real and fake images
- Train $\mathcal{G}$ (but not $\mathcal{D}$) on fake images
- Repeat

Loss usually does not converge

- Look at generated images during training
- Accuracy of $\mathcal{D}$ should eventually be ≈ 0.5

Typical training loop (many variants exist)

- k is a hyperparameter (often $k = 1$ works fine)

```
while not done:
  for k iterations:
    sample half of mini-batch from dataset (class real)
    sample half of mini-batch from G (class fake)
    train D on mini-batch (but not G)

  sample mini-batch from G (class real (!))
  train G trough D on mini-batch (but not D itself)
```

GANs

GANs are a flexible framework with many applications

- More in next lecture

Make sure to pre-train both $\mathcal{G}$ and $\mathcal{D}$ (if possible)

- Training GANs from scratch is a pain

Bibliography

[1] Noh et al. Deconvolution Networks. 2015

[2] Ronneberger et al. U-Net. 2015

[3] Cao et al. OpenPose. 2018

[4] Shi et al. Subpixel Convolutions. 2016