Carnegie Mellon University

Recitation 0.25

How to read research papers?

Introduction to Deep Learning (11-485/685/785)

By Prof. Bhiksha Raj

TA: Ishita Gupta

Table of Contents

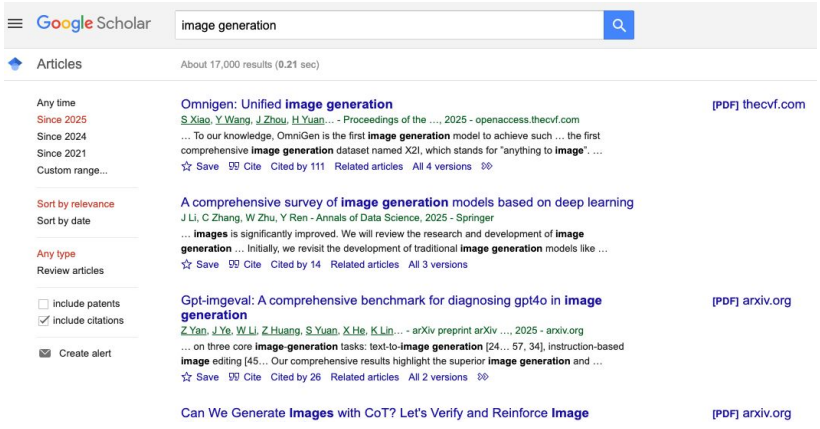
- Motivation
- Where to find papers?
- How to decide which papers to read?
- Example walk-through

Motivation

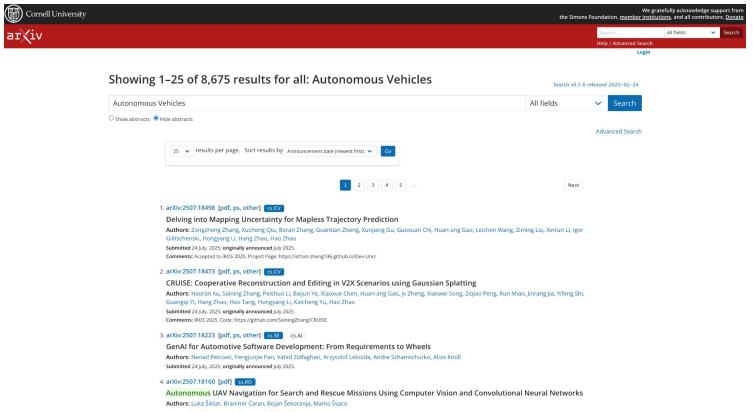
- Reading research papers efficiently is a must-have skill to stay up-to-date with cutting-edge research; especially if you're aiming to get into ML research or engineering roles.
- We will be referencing a lot of papers in our lectures, quizzes, and homeworks.
- For students enrolled in 685 or 785 - you will **need** to read multiple research papers to understand the state-of-the-art, select baselines, and to come up with novel ideas for your own project!

Make sure to revisit this recitation once you start working on your projects :-)

Where to find Research Papers?

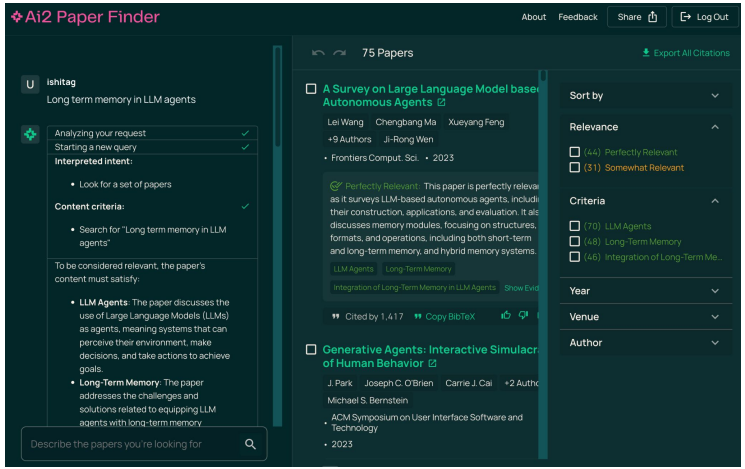


https://scholar.google.com/

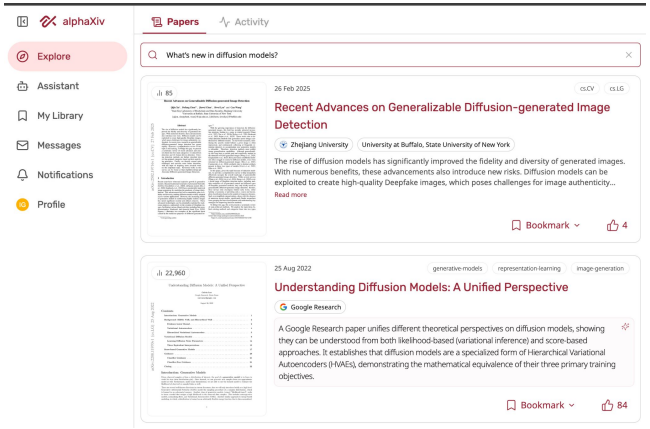


https://arxiv.org/

Where to find Research Papers?

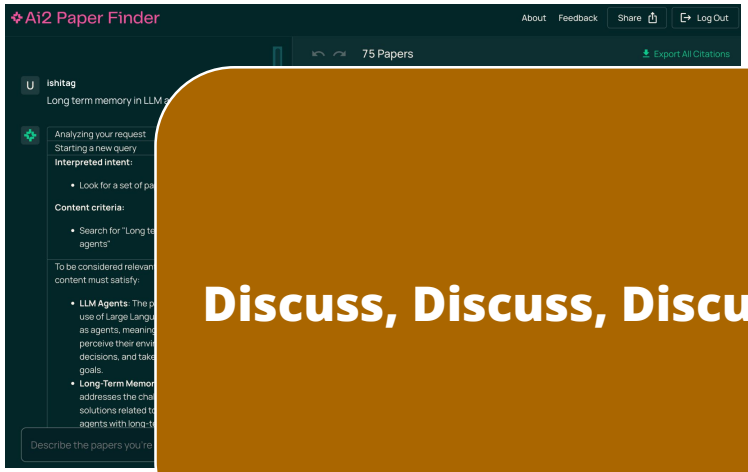


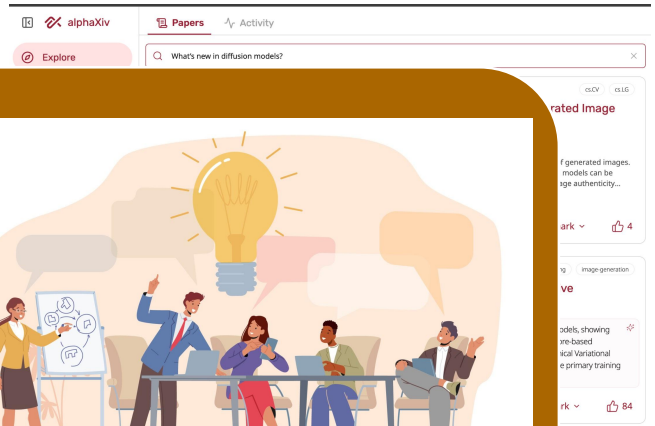
https://paperfinder.allen.ai/



https://www.alphaxiv.org/

Where to find Research Papers?





Discuss, Discuss, Discuss

https://p

How to Decide which Papers to Read ?

- **Start with a list** of papers, (6-10) on a topic of your interest.
- **Skim** through each one of them:
  - Abstract, Introduction, Conclusion, and Figures
    - *"What is the **purpose** of this paper?"*
    - *"What **problem** are they trying to solve?"*
    - "*Are there any **contributions** that stand out?"*
    - Bonus: Use AI tools, if needed, to get key contributions or summaries of papers!
- **Eliminate**
- **Narrow** it down to 2-3.

Stage 1

- Title, Authors, Organizations

PixelNet: Representation *of* the pixels, *by* the pixels, and *for* the pixels.

Aayush Bansal[1] Xinlei Chen[1] Bryan Russell[2] Abhinav Gupta[1] Deva Ramanan[1]
[1]Carnegie Mellon University [2]Adobe Research
http://www.cs.cmu.edu/~aayushb/pixelNet/

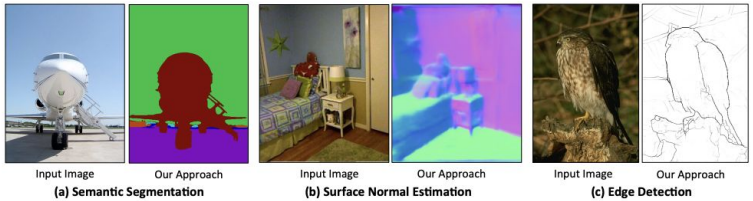


Figure 1. Our framework applied to three different pixel prediction problems with minor modification of the architecture (last layer) and training process (epochs). Note how our approach recovers the fine details for segmentation (left), surface normal (middle), and semantic boundaries for edge detection (right).

Abstract

We explore design principles for general pixel-level prediction problems, from low-level edge detection to mid-level surface normal estimation to high-level semantic segmentation. Convolutional predictors, such as the fully-convolutional network (FCN), have achieved remarkable success by exploiting the spatial redundancy of neighboring pixels through convolutional processing. Though computationally efficient, we point out that such approaches are not statistically efficient during learning precisely because spatial redundancy limits the information learned from neighboring pixels. We demonstrate that stratified sampling of pixels allows one to (1) add diversity during batch updates, speeding up learning; (2) explore complex nonlinear predictors, improving accuracy; and (3) efficiently train state-of-the-art models tabula rasa *(i.e., "from scratch") for diverse pixel-labeling tasks. Our single architecture produces state-of-the-art results for semantic segmentation on PASCAL-Context dataset, surface normal estimation on NYUDv2 depth dataset, and edge detection on BSDS.*

1. Introduction

A number of computer vision problems can be formulated as a dense pixel-wise prediction problem. These include **low-level** tasks such as edge detection [21, 64, 94] and optical flow [5, 30, 86], **mid-level** tasks such as depth/normal recovery [6, 24, 25, 81, 89], and **high-level** tasks such as keypoint prediction [13, 36, 78, 91], object detection [43], and semantic segmentation [16, 27, 40, 62, 67, 82]. Though such a formulation is attractive because of its generality, one obvious difficulty is the enormous associated output space. For example, a 100×100 image with 10 discrete class labels per pixel yields an output label space of size 10^5. One strategy is to treat this as a *spatially-invariant label prediction* problem, where one predicts a separate label per pixel using a convolutional architecture. Neural networks with convolutional output predictions, also called Fully Convolutional Networks (FCNs) [16, 62, 65, 77], appear to be a promising architecture in this direction.

But is this the ideal formulation of dense pixel-labeling? While *computationally efficient* for generating predictions at test time, we argue that it is *not statistically efficient* for gradient-based learning. Stochastic gradient descent (SGD) assumes that training data are sampled independently and from an identical distribution (*i.i.d.*) [11]. Indeed, a commonly-used heuristic to ensure approximately *i.i.d.* samples is random permutation of the training data, which can significantly improve learnability [56]. It is well known that pixels in a given image are highly correlated and not independent [45]. Following this observation, one might be tempted to randomly permute pixels during learning, but this destroys the spatial regularity that convolutional architectures so cleverly exploit! In this paper, we explore the

Stage 1

- Title, Authors, Organizations
- Website

PixelNet: Representation *of* the pixels, *by* the pixels, and *for* the pixels.

Aayush Bansal[1] Xinlei Chen[1] Bryan Russell[2] Abhinav Gupta[1] Deva Ramanan[1]
[1]Carnegie Mellon University [2]Adobe Research

http://www.cs.cmu.edu/~aayushb/pixelNet/

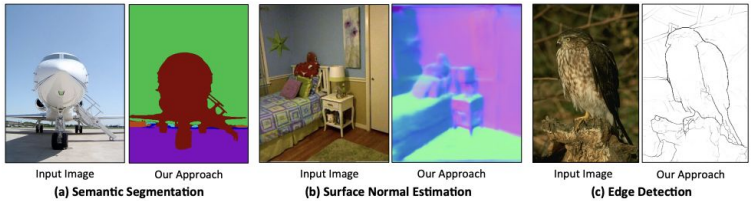


Figure 1. Our framework applied to three different pixel prediction problems with minor modification of the architecture (last layer) and training process (epochs). Note how our approach recovers the fine details for segmentation (left), surface normal (middle), and semantic boundaries for edge detection (right).

Abstract

We explore design principles for general pixel-level prediction problems, from low-level edge detection to mid-level surface normal estimation to high-level semantic segmentation. Convolutional predictors, such as the fully-convolutional network (FCN), have achieved remarkable success by exploiting the spatial redundancy of neighboring pixels through convolutional processing. Though computationally efficient, we point out that such approaches are not statistically efficient during learning precisely because spatial redundancy limits the information learned from neighboring pixels. We demonstrate that stratified sampling of pixels allows one to (1) add diversity during batch updates, speeding up learning; (2) explore complex nonlinear predictors, improving accuracy; and (3) efficiently train state-of-the-art models tabula rasa *(i.e., "from scratch") for diverse pixel-labeling tasks. Our single architecture produces state-of-the-art results for semantic segmentation on PASCAL-Context dataset, surface normal estimation on NYUDv2 depth dataset, and edge detection on BSDS.*

1. Introduction

A number of computer vision problems can be formulated as a dense pixel-wise prediction problem. These include **low-level** tasks such as edge detection [21, 64, 94] and optical flow [5, 30, 86], **mid-level** tasks such as depth/normal recovery [6, 24, 25, 81, 89], and **high-level** tasks such as keypoint prediction [13, 36, 78, 91], object detection [43], and semantic segmentation [16, 27, 40, 62, 67, 82]. Though such a formulation is attractive because of its generality, one obvious difficulty is the enormous associated output space. For example, a 100×100 image with 10 discrete class labels per pixel yields an output label space of size 10^5. One strategy is to treat this as a *spatially-invariant label prediction* problem, where one predicts a separate label per pixel using a convolutional architecture. Neural networks with convolutional output predictions, also called Fully Convolutional Networks (FCNs) [16, 62, 65, 77], appear to be a promising architecture in this direction.

But is this the ideal formulation of dense pixel-labeling? While *computationally efficient* for generating predictions at test time, we argue that it is *not statistically efficient* for gradient-based learning. Stochastic gradient descent (SGD) assumes that training data are sampled independently and from an identical distribution (*i.i.d.*) [11]. Indeed, a commonly-used heuristic to ensure approximately *i.i.d.* samples is random permutation of the training data, which can significantly improve learnability [56]. It is well known that pixels in a given image are highly correlated and not independent [45]. Following this observation, one might be tempted to randomly permute pixels during learning, but this destroys the spatial regularity that convolutional architectures so cleverly exploit! In this paper, we explore the

Stage 1

- Title, Authors, Organizations
- Website
- Key Results

PixelNet: Representation *of* the pixels, *by* the pixels, and *for* the pixels.

Aayush Bansal[1] Xinlei Chen[1] Bryan Russell[2] Abhinav Gupta[1] Deva Ramanan[1]
[1]Carnegie Mellon University [2]Adobe Research

http://www.cs.cmu.edu/~aayushb/pixelNet/

Figure 1. Our framework applied to three different pixel prediction problems with minor modification of the architecture (last layer) and training process (epochs). Note how our approach recovers the fine details for segmentation (left), surface normal (middle), and semantic boundaries for edge detection (right).

Abstract

We explore design principles for general pixel-level prediction problems, from low-level edge detection to mid-level surface normal estimation to high-level semantic segmentation. Convolutional predictors, such as the fully-convolutional network (FCN), have achieved remarkable success by exploiting the spatial redundancy of neighboring pixels through convolutional processing. Though computationally efficient, we point out that such approaches are not statistically efficient during learning precisely because spatial redundancy limits the information learned from neighboring pixels. We demonstrate that stratified sampling of pixels allows one to (1) add diversity during batch updates, speeding up learning; (2) explore complex nonlinear predictors, improving accuracy; and (3) efficiently train state-of-the-art models tabula rasa *(i.e., "from scratch") for diverse pixel-labeling tasks. Our single architecture produces state-of-the-art results for semantic segmentation on PASCAL-Context dataset, surface normal estimation on NYUDv2 depth dataset, and edge detection on BSDS.*

1. Introduction

A number of computer vision problems can be formulated as a dense pixel-wise prediction problem. These include **low-level** tasks such as edge detection [21, 64, 94] and optical flow [5, 30, 86], **mid-level** tasks such as depth/normal recovery [6, 24, 25, 81, 89], and **high-level** tasks such as keypoint prediction [13, 36, 78, 91], object detection [43], and semantic segmentation [16, 27, 40, 62, 67, 82]. Though such a formulation is attractive because of its generality, one obvious difficulty is the enormous associated output space. For example, a 100×100 image with 10 discrete class labels per pixel yields an output label space of size 10^5. One strategy is to treat this as a *spatially-invariant label prediction* problem, where one predicts a separate label per pixel using a convolutional architecture. Neural networks with convolutional output predictions, also called Fully Convolutional Networks (FCNs) [16, 62, 65, 77], appear to be a promising architecture in this direction.

But is this the ideal formulation of dense pixel-labeling? While *computationally efficient* for generating predictions at test time, we argue that it is *not statistically efficient* for gradient-based learning. Stochastic gradient descent (SGD) assumes that training data are sampled independently and from an identical distribution (*i.i.d.*) [11]. Indeed, a commonly-used heuristic to ensure approximately *i.i.d.* samples is random permutation of the training data, which can significantly improve learnability [56]. It is well known that pixels in a given image are highly correlated and not independent [45]. Following this observation, one might be tempted to randomly permute pixels during learning, but this destroys the spatial regularity that convolutional architectures so cleverly exploit! In this paper, we explore the

Stage 1

- Title, Authors, Organizations
- Website
- Key Results
- Abstract

PixelNet: Representation *of* the pixels, *by* the pixels, and *for* the pixels.

Aayush Bansal[1] Xinlei Chen[1] Bryan Russell[2] Abhinav Gupta[1] Deva Ramanan[1]
[1]Carnegie Mellon University [2]Adobe Research

http://www.cs.cmu.edu/~aayushb/pixelNet/

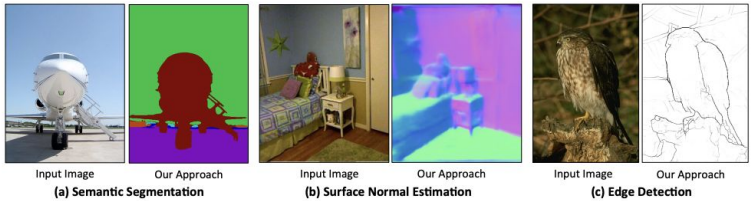


Figure 1. Our framework applied to three different pixel prediction problems with minor modification of the architecture (last layer) and training process (epochs). Note how our approach recovers the fine details for segmentation (left), surface normal (middle), and semantic boundaries for edge detection (right).

Abstract

We explore design principles for general pixel-level prediction problems, from low-level edge detection to mid-level surface normal estimation to high-level semantic segmentation. Convolutional predictors, such as the fully-convolutional network (FCN), have achieved remarkable success by exploiting the spatial redundancy of neighboring pixels through convolutional processing. Though computationally efficient, we point out that such approaches are not statistically efficient during learning precisely because spatial redundancy limits the information learned from neighboring pixels. We demonstrate that stratified sampling of pixels allows one to (1) add diversity during batch updates, speeding up learning; (2) explore complex nonlinear predictors, improving accuracy; and (3) efficiently train state-of-the-art models tabula rasa *(i.e., "from scratch") for diverse pixel-labeling tasks. Our single architecture produces state-of-the-art results for semantic segmentation on PASCAL-Context dataset, surface normal estimation on NYUDv2 depth dataset, and edge detection on BSDS.*

1. Introduction

A number of computer vision problems can be formulated as a dense pixel-wise prediction problem. These include **low-level** tasks such as edge detection [21, 64, 94] and optical flow [5, 30, 86], **mid-level** tasks such as depth/normal recovery [6, 24, 25, 81, 89], and **high-level** tasks such as keypoint prediction [13, 36, 78, 91], object detection [43], and semantic segmentation [16, 27, 40, 62, 67, 82]. Though such a formulation is attractive because of its generality, one obvious difficulty is the enormous associated output space. For example, a 100×100 image with 10 discrete class labels per pixel yields an output label space of size 10^5. One strategy is to treat this as a *spatially-invariant label prediction* problem, where one predicts a separate label per pixel using a convolutional architecture. Neural networks with convolutional output predictions, also called Fully Convolutional Networks (FCNs) [16, 62, 65, 77], appear to be a promising architecture in this direction.

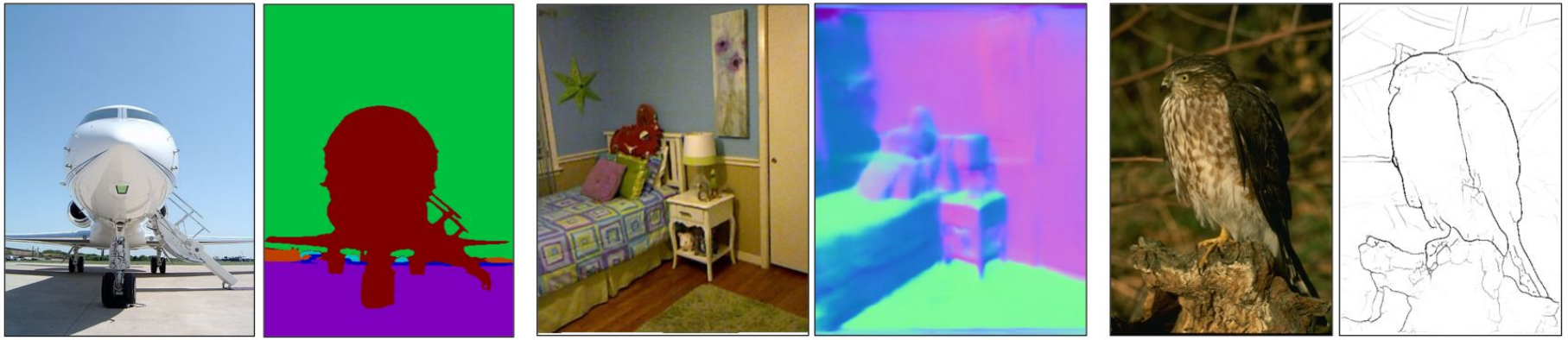
But is this the ideal formulation of dense pixel-labeling? While *computationally efficient* for generating predictions at test time, we argue that it is *not statistically efficient* for gradient-based learning. Stochastic gradient descent (SGD) assumes that training data are sampled independently and from an identical distribution (*i.i.d.*) [11]. Indeed, a commonly-used heuristic to ensure approximately *i.i.d.* samples is random permutation of the training data, which can significantly improve learnability [56]. It is well known that pixels in a given image are highly correlated and not independent [45]. Following this observation, one might be tempted to randomly permute pixels during learning, but this destroys the spatial regularity that convolutional architectures so cleverly exploit! In this paper, we explore the

PixelNet: Representation *of* the pixels, *by* the pixels, and *for* the pixels.

Aayush Bansal[1] Xinlei Chen[1] Bryan Russell[2] Abhinav Gupta[1] Deva Ramanan[1]
[1]Carnegie Mellon University [2]Adobe Research

http://www.cs.cmu.edu/~aayushb/pixelNet/

https://www.aayushbansal.xyz/pixelNet/



(a) Semantic Segmentation **(b) Surface Normal Estimation** **(c) Edge Detection**

Figure 1. Our framework applied to three different pixel prediction problems with minor modification of the architecture (last layer) and training process (epochs). Note how our approach recovers the fine details for segmentation (left), surface normal (middle), and semantic boundaries for edge detection (right).

Abstract

We explore design principles for general pixel-level prediction problems, from low-level edge detection to mid-level surface normal estimation to high-level semantic segmentation. Convolutional predictors, such as the fully-convolutional network (FCN), have achieved remarkable success by exploiting the spatial redundancy of neighboring pixels through convolutional processing. Though computationally efficient, we point out that such approaches are not statistically efficient during learning precisely because spatial redundancy limits the information learned from neighboring pixels. We demonstrate that stratified sampling of pixels allows one to (1) add diversity during batch updates, speeding up learning; (2) explore complex nonlinear predictors, improving accuracy; and (3) efficiently train state-of-the-art models tabula rasa *(i.e., "from scratch") for diverse pixel-labeling tasks. Our single architecture produces state-of-the-art results for semantic segmentation on PASCAL-Context dataset, surface normal estimation on NYUDv2 depth dataset, and edge detection on BSDS.*

Stage 2

- Introduction

Abstract

We explore design principles for general pixel-level prediction problems, from low-level edge detection to mid-level surface normal estimation to high-level semantic segmentation. Convolutional predictors, such as the fully-convolutional network (FCN), have achieved remarkable success by exploiting the spatial redundancy of neighboring pixels through convolutional processing. Though computationally efficient, we point out that such approaches are not statistically efficient during learning precisely because spatial redundancy limits the information learned from neighboring pixels. We demonstrate that stratified sampling of pixels allows one to (1) add diversity during batch updates, speeding up learning; (2) explore complex nonlinear predictors, improving accuracy; and (3) efficiently train state-of-the-art models tabula rasa *(i.e., "from scratch") for diverse pixel-labeling tasks. Our single architecture produces state-of-the-art results for semantic segmentation on PASCAL-Context dataset, surface normal estimation on NYUDv2 depth dataset, and edge detection on BSDS.*

1. Introduction

A number of computer vision problems can be formulated as a dense pixel-wise prediction problem. These include **low-level** tasks such as edge detection [21, 64, 94] and optical flow [5, 30, 86], **mid-level** tasks such as depth/normal recovery [6, 24, 25, 81, 89], and **high-level** tasks such as keypoint prediction [13, 36, 78, 91], object detection [43], and semantic segmentation [16, 27, 40, 62, 67, 82]. Though such a formulation is attractive because of its generality, one obvious difficulty is the enormous associated output space. For example, a 100×100 image with 10 discrete class labels per pixel yields an output label space of size 10^5. One strategy is to treat this as a *spatially-invariant label prediction* problem, where one predicts a separate label per pixel using a convolutional architecture. Neural networks with convolutional output predictions, also called Fully Convolutional Networks (FCNs) [16, 62, 65, 77], appear to be a promising architecture in this direction.

But is this the ideal formulation of dense pixel-labeling? While *computationally efficient* for generating predictions at test time, we argue that it is *not statistically efficient* for gradient-based learning. Stochastic gradient descent (SGD) assumes that training data are sampled independently and from an identical distribution (*i.i.d.*) [11]. Indeed, a commonly-used heuristic to ensure approximately *i.i.d.* samples is random permutation of the training data, which can significantly improve learnability [56]. It is well known that pixels in a given image are highly correlated and not independent [45]. Following this observation, one might be tempted to randomly permute pixels during learning, but this destroys the spatial regularity that convolutional architectures so cleverly exploit! In this paper, we explore the

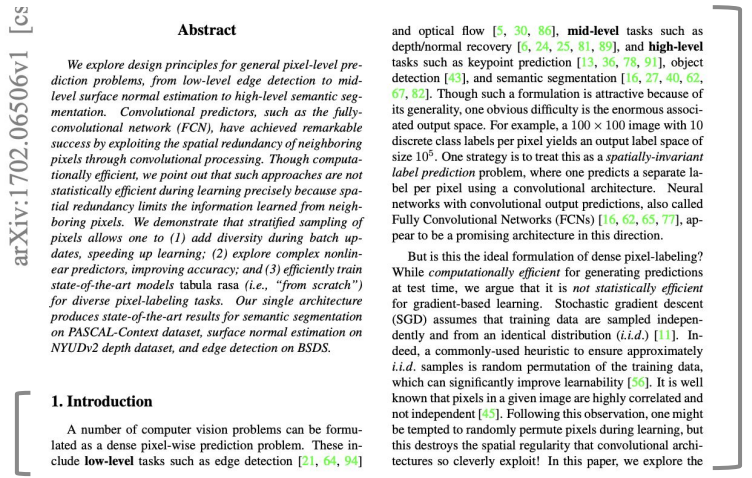
formance between uniform and biased sampling. Clearly biased sampling toward positives can help the performance. Qualitatively, we find our network tends to have better results for semantic-contours (*e.g.* around an object), particularly after including *conv-7* features. Figure 9 shows some qualitative results comparing our network with the HED model. Interestingly, our model explicitly removed the edges inside the *zebra*, but when the model cannot recognize it (*e.g.* its head is out of the picture), it still marks the edges on the black-and-white stripes. Our model appears to be making use of more higher-level information than past work on edge detection. More analysis and details are in Appendix C.

6. Discussion

We have described a convolutional pixel-level architecture that, with minor modifications, produces state-of-the-art accuracy on diverse high-level, mid-level, and low-level tasks. We demonstrate results on highly-benchmarked semantic segmentation, surface normal estimation, and edge detection datasets. Our results are made possible by careful analysis of computational and statistical considerations associated with convolutional predictors. Convolution exploits spatial redundancy of pixel neighborhoods for efficient computation, but this redundancy also impedes learning. We propose a simple solution based on stratified sam-

Stage 2

- Introduction
- Conclusion/Discussion



Abstract

We explore design principles for general pixel-level prediction problems, from low-level edge detection to mid-level surface normal estimation to high-level semantic segmentation. Convolutional predictors, such as the fully-convolutional network (FCN), have achieved remarkable success by exploiting the spatial redundancy of neighboring pixels through convolutional processing. Though computationally efficient, we point out that such approaches are not statistically efficient during learning precisely because spatial redundancy limits the information learned from neighboring pixels. We demonstrate that stratified sampling of pixels allows one to (1) add diversity during batch updates, speeding up learning; (2) explore complex nonlinear predictors, improving accuracy; and (3) efficiently train state-of-the-art models tabula rasa *(i.e., "from scratch") for diverse pixel-labeling tasks. Our single architecture produces state-of-the-art results for semantic segmentation on PASCAL-Context dataset, surface normal estimation on NYUDv2 depth dataset, and edge detection on BSDS.*

1. Introduction

A number of computer vision problems can be formulated as a dense pixel-wise prediction problem. These include **low-level** tasks such as edge detection [21, 64, 94] and optical flow [5, 30, 86], **mid-level** tasks such as depth/normal recovery [6, 24, 25, 81, 89], and **high-level** tasks such as keypoint prediction [13, 36, 78, 91], object detection [43], and semantic segmentation [16, 27, 40, 62, 67, 82]. Though such a formulation is attractive because of its generality, one obvious difficulty is the enormous associated output space. For example, a 100×100 image with 10 discrete class labels per pixel yields an output label space of size 10^5. One strategy is to treat this as a *spatially-invariant label prediction* problem, where one predicts a separate label per pixel using a convolutional architecture. Neural networks with convolutional output predictions, also called Fully Convolutional Networks (FCNs) [16, 62, 65, 77], appear to be a promising architecture in this direction.

But is this the ideal formulation of dense pixel-labeling? While *computationally efficient* for generating predictions at test time, we argue that it is *not statistically efficient* for gradient-based learning. Stochastic gradient descent (SGD) assumes that training data are sampled independently and from an identical distribution (*i.i.d.*) [11]. Indeed, a commonly-used heuristic to ensure approximately *i.i.d.* samples is random permutation of the training data, which can significantly improve learnability [56]. It is well known that pixels in a given image are highly correlated and not independent [45]. Following this observation, one might be tempted to randomly permute pixels during learning, but this destroys the spatial regularity that convolutional architectures so cleverly exploit! In this paper, we explore the

formance between uniform and biased sampling. Clearly biased sampling toward positives can help the performance. Qualitatively, we find our network tends to have better results for semantic-contours (*e.g.* around an object), particularly after including *conv-7* features. Figure 9 shows some qualitative results comparing our network with the HED model. Interestingly, our model explicitly removed the edges inside the *zebra*, but when the model cannot recognize it (*e.g.* its head is out of the picture), it still marks the edges on the black-and-white stripes. Our model appears to be making use of more higher-level information than past work on edge detection. More analysis and details are in Appendix C.

6. Discussion

We have described a convolutional pixel-level architecture that, with minor modifications, produces state-of-the-art accuracy on diverse high-level, mid-level, and low-level tasks. We demonstrate results on highly-benchmarked semantic segmentation, surface normal estimation, and edge detection datasets. Our results are made possible by careful analysis of computational and statistical considerations associated with convolutional predictors. Convolution exploits spatial redundancy of pixel neighborhoods for efficient computation, but this redundancy also impedes learning. We propose a simple solution based on stratified sam-

1. Introduction

A number of computer vision problems can be formulated as a dense pixel-wise prediction problem. These include **low-level** tasks such as edge detection [21, 64, 94] and optical flow [5, 30, 86], **mid-level** tasks such as depth/normal recovery [6, 24, 25, 81, 89], and **high-level** tasks such as keypoint prediction [13, 36, 78, 91], object detection [43], and semantic segmentation [16, 27, 40, 62, 67, 82]. Though such a formulation is attractive because of its generality, one obvious difficulty is the enormous associated output space. For example, a 100×100 image with 10 discrete class labels per pixel yields an output label space of size 10^5. One strategy is to treat this as a *spatially-invariant label prediction* problem, where one predicts a separate label per pixel using a convolutional architecture. Neural networks with convolutional output predictions, also called Fully Convolutional Networks (FCNs) [16, 62, 65, 77], appear to be a promising architecture in this direction.

But is this the ideal formulation of dense pixel-labeling? While *computationally efficient* for generating predictions at test time, we argue that it is *not statistically efficient* for gradient-based learning. Stochastic gradient descent (SGD) assumes that training data are sampled independently and from an identical distribution (*i.i.d.*) [11]. Indeed, a commonly-used heuristic to ensure approximately *i.i.d.* samples is random permutation of the training data, which can significantly improve learnability [56]. It is well known that pixels in a given image are highly correlated and not independent [45]. Following this observation, one might be tempted to randomly permute pixels during learning, but this destroys the spatial regularity that convolutional architectures so cleverly exploit! In this paper, we explore the tradeoff between statistical and computational efficiency for convolutional learning, and investigate simply *sampling* a modest number of pixels across a small number of images for each SGD batch update, exploiting convolutional processing where possible.

Contributions: (1) We experimentally validate that, thanks to spatial correlations between pixels, just sampling a small number of pixels per image is sufficient for learning. More importantly, sampling allows us to train end-to-end particular non-linear models not earlier possible, and explore several avenues for improving both the efficiency and performance of FCN-based architectures. (2) In contrast to the vast majority of models that make use of pre-trained networks, we show that pixel-level optimization can be used to train models *tabula rasa*, or "from scratch" with simple random Gaussian initialization. Intuitively, pixel-level labels provide a large amount of supervision compared to image-level labels, given proper accounting of correlations. Without using any extra data, our model outperforms previous unsupervised/self-supervised approaches for semantic segmentation on PASCAL VOC-2012 [26], and is competitive to fine-tuning from pre-trained models for surface normal estimation. (3). Using a single architecture and without much modification in parameters, we show state-of-the-art performance for edge detection on BSDS [4], surface normal estimation on NYUDv2 depth dataset [83], and semantic segmentation on the PASCAL-Context dataset [68].

2. Background

- Conclusion/Discussion:
  - Results ✓
  - Limitations ✓
  - Future work ✓

6. Discussion

We have described a convolutional pixel-level architecture that, with minor modifications, produces state-of-the-art accuracy on diverse high-level, mid-level, and low-level tasks. We demonstrate results on highly-benchmarked semantic segmentation, surface normal estimation, and edge detection datasets. Our results are made possible by careful analysis of computational and statistical considerations associated with convolutional predictors. Convolution exploits spatial redundancy of pixel neighborhoods for efficient computation, but this redundancy also impedes learning. We propose a simple solution based on stratified sampling that injects diversity while taking advantage of amortized convolutional processing. Finally, our efficient learning scheme allow us to explore nonlinear functions of multi-scale features that encode both high-level context and low-level spatial detail, which appears relevant for most pixel prediction tasks.

Stage 3: Read but (freely) skip the math equations

- Figures

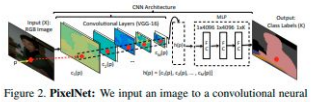


Figure 2. **PixelNet:** We input an image to a convolutional neural network, and extract hypercolumn descriptor for a sampled pixel from multiple convolutional layers. The hypercolumn descriptor is then fed to a multi-layer perceptron (MLP) for the non-linear optimization, and the last layer of MLP outputs the required response for the task. See text for more details about the use of network at training/test time.

linear g, which in turn significantly boosts performance. Our insight is inspired by past approaches that use sampling to training networks for surface normal estimation [6] and image colorization [55], though we focus on general design principles by analyzing the impact of sampling for efficiency, accuracy, and *tabula rasa* learning for diverse tasks.

Accelerating SGD: There exists a large literature on accelerating stochastic gradient descent. We refer the reader to [11] for an excellent introduction. Though naturally a sequential algorithm that processes one data example at a time, much recent work focuses on mini-batch methods that can exploit parallelism in GPU architectures [18] or clusters [18]. One general theme is efficient online approximation of second-order methods [10], which can model correlations between input features. Batch normalization [46] computes correlation statistics between samples in a batch, producing noticeable improvements in convergence speed. Our work builds similar insights directly into convolutional networks without explicit second-order statistics.

3. PixelNet

This section describes our approach for pixel-wise prediction, making use of the notation introduced in the previous section. We first formalize our pixelwise prediction architecture, and then discuss statistically efficient mini-batch training.

Architecture: As in past work, our architecture makes use of multiscale convolutional features, which we write as a hypercolumn descriptor:

$$h_p = [c_1(p), c_2(p), \ldots, c_M(p)]$$

We learn a nonlinear predictor $f_{\theta,p} = g(h_p)$ implemented as a multi-layer perceptron (MLP) [9] defined over hypercolumn features. We use a MLP, which can be implemented as a series of "fully-connected" layers followed by ReLU activation functions. Importantly, the last layer must be of

Method	IoU_1 (V)	IoU_2 (V)	Mean	Median	RMSE	11.25°	22.5°	30°
$1 \times 40,000$	7.9	15.5	24.8	19.5	31.6	29.7	56.1	68.5
$5 \times 2,000$	**38.4**	**47.9**	**23.4**	**17.2**	**30.5**	**33.9**	**60.6**	**71.8**

Table 4. **Statistical Diversity Matters:** For a given computational budget, using diverse set of pixels from more images shows better performance over more pixels from a few images. IoU_1 and IoU_2 show performance for 10K and 20K iterations of SGD for semantic segmentation. For surface normal estimation, we show performance for 10K iterations of SGD. This suggest that sampling leads to faster convergence.

classifier is used.

4.3. Training from scratch

Prevailing methods for training deep models make use of a pre-trained (e.g., ImageNet [79]) model as initialization for fine-tuning for the task at hand. Most network architectures (including ours) improve in performance with pre-trained models. A major concern is the availability of sufficient data to train deep models for pixel-level prediction problems. However, because our optimization is based on randomly-sampled pixels instead of images, there is potentially more unique data available for SGD to learn a model from a random initialization. We show how *sampling* and *batch-normalization* enables models to be trained from scratch. This enables our networks to be used in problems with limited training data, or where natural image data does not apply (e.g., molecular biology, tissue segmentation etc. [70, 95]). We will show that our results also have implications for unsupervised representation learning [1, 20, 23, 34, 37, 48, 57, 55, 66, 72, 73, 74, 76, 90, 99, 100].

Random Initialization: We randomly initialize the parameters of a VGG-16 network from a Gaussian distribution. Training a VGG-16 network architecture is not straight forward, and required stage-wise training for the image classification task [84]. It seems daunting to train such a model from scratch for a pixel-level task where we want to learn both coarse and fine information. In our experiments, we found batch normalization to be an effective tool for converging a model trained from scratch.

We train the models for semantic segmentation and surface normal estimation. The middle-row in Table 5 shows the performance for semantic segmentation and surface normal estimation trained from scratch. The model trained from scratch for surface normal estimation is within 2-3% of current state-of-the-art performing method. The model for semantic segmentation achieves 48.7% on PASCAL VOC-2012 test set when trained from scratch. To the best of our knowledge, these are the best numbers reported on these two tasks when trained from scratch, and exceeds the performance of other unsupervised/self-supervised approaches [20, 55, 74, 90, 99] that required extra ImageNet data [79].

Stage 3: Read but (freely) skip the math equations

- Figures
- Methodology & Experiments

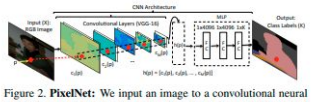


Figure 2. **PixelNet:** We input an image to a convolutional neural network, and extract hypercolumn descriptor for a sampled pixel from multiple convolutional layers. The hypercolumn descriptor is then fed to a multi-layer perceptron (MLP) for the non-linear optimization, and the last layer of MLP outputs the required response for the task. See text for more details about the use of network at training/test time.

linear g, which in turn significantly boosts performance. Our insight is inspired by past approaches that use sampling to training networks for surface normal estimation [6] and image colorization [55], though we focus on general design principles by analyzing the impact of sampling for efficiency, accuracy, and *tabula rasa* learning for diverse tasks.

Accelerating SGD: There exists a large literature on accelerating stochastic gradient descent. We refer the reader to [11] for an excellent introduction. Though naturally a sequential algorithm that processes one data example at a time, much recent work focuses on mini-batch methods that can exploit parallelism in GPU architectures [18] or clusters [18]. One general theme is efficient online approximation of second-order methods [10], which can model correlations between input features. Batch normalization [46] computes correlation statistics between samples in a batch, producing noticeable improvements in convergence speed. Our work builds similar insights directly into convolutional networks without explicit second-order statistics.

3. PixelNet

This section describes our approach for pixel-wise prediction, making use of the notation introduced in the previous section. We first formalize our pixelwise prediction architecture, and then discuss statistically efficient mini-batch training.

Architecture: As in past work, our architecture makes use of multiscale convolutional features, which we write as a hypercolumn descriptor:

$$h_p = [c_1(p), c_2(p), \ldots, c_M(p)]$$

We learn a nonlinear predictor $f_{\theta,p} = g(h_p)$ implemented as a multi-layer perceptron (MLP) [9] defined over hypercolumn features. We use a MLP, which can be implemented as a series of "fully-connected" layers followed by ReLU activation functions. Importantly, the last layer must be of

Method	IoU_1 (V)	IoU_2 (V)	Mean	Median	RMSE	11.25°	22.5°	30°
$1 \times 40,000$	7.9	15.5	24.8	19.5	31.6	29.7	56.1	68.5
$5 \times 2,000$	**38.4**	**47.9**	**23.4**	**17.2**	**30.5**	**33.9**	**60.6**	**71.8**

Table 4. **Statistical Diversity Matters:** For a given computational budget, using diverse set of pixels from more images shows better performance over more pixels from a few images. IoU_1 and IoU_2 show performance for 10K and 20K iterations of SGD for semantic segmentation. For surface normal estimation, we show performance for 10K iterations of SGD. This suggest that sampling leads to faster convergence.

classifier is used.

4.3. Training from scratch

Prevailing methods for training deep models make use of a pre-trained (e.g., ImageNet [79]) model as initialization for fine-tuning for the task at hand. Most network architectures (including ours) improve in performance with pre-trained models. A major concern is the availability of sufficient data to train deep models for pixel-level prediction problems. However, because our optimization is based on randomly-sampled pixels instead of images, there is potentially more unique data available for SGD to learn a model from a random initialization. We show how *sampling* and *batch-normalization* enables models to be trained from scratch. This enables our networks to be used in problems with limited training data, or where natural image data does not apply (e.g., molecular biology, tissue segmentation etc. [70, 95]). We will show that our results also have implications for unsupervised representation learning [1, 20, 23, 34, 37, 48, 57, 55, 66, 72, 73, 74, 76, 90, 99, 100].

Random Initialization: We randomly initialize the parameters of a VGG-16 network from a Gaussian distribution. Training a VGG-16 network architecture is not straight forward, and required stage-wise training for the image classification task [84]. It seems daunting to train such a model from scratch for a pixel-level task where we want to learn both coarse and fine information. In our experiments, we found batch normalization to be an effective tool for converging a model trained from scratch.

We train the models for semantic segmentation and surface normal estimation. The middle-row in Table 5 shows the performance for semantic segmentation and surface normal estimation trained from scratch. The model trained from scratch for surface normal estimation is within 2-3% of current state-of-the-art performing method. The model for semantic segmentation achieves 48.7% on PASCAL VOC-2012 test set when trained from scratch. To the best of our knowledge, these are the best numbers reported on these two tasks when trained from scratch, and exceeds the performance of other unsupervised/self-supervised approaches [20, 55, 74, 90, 99] that required extra ImageNet data [79].

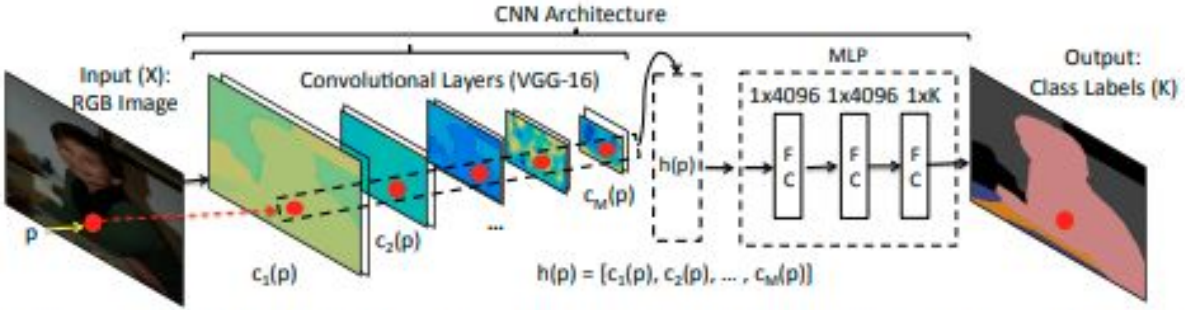


Figure 2. **PixelNet:** We input an image to a convolutional neural network, and extract hypercolumn descriptor for a sampled pixel from multiple convolutional layers. The hypercolumn descriptor is then fed to a multi-layer perceptron (MLP) for the non-linear optimization, and the last layer of MLP outputs the required response for the task. See text for more details about the use of network at training/test time.

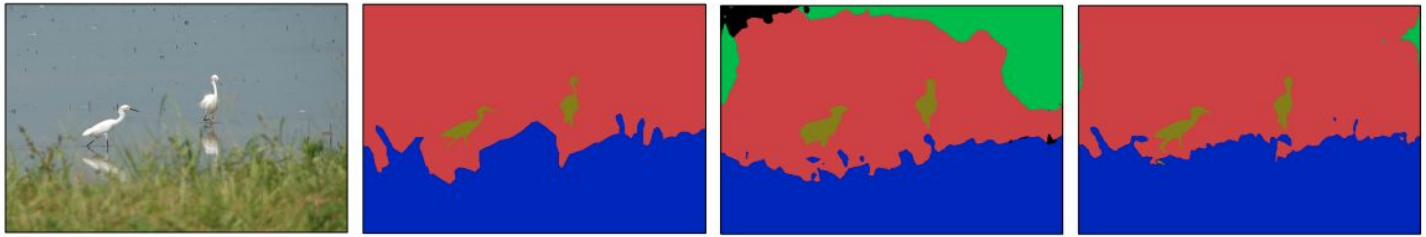


Figure 4. Segmentation results on PASCAL-Context 59-class. Our approach uses an MLP to integrate information from both lower (*e.g.* 1_2) and higher (*e.g. conv-7*) layers, which allows us to better capture both global structure (object/scene layout) and fine details (small objects) compared to *FCN-8s*.

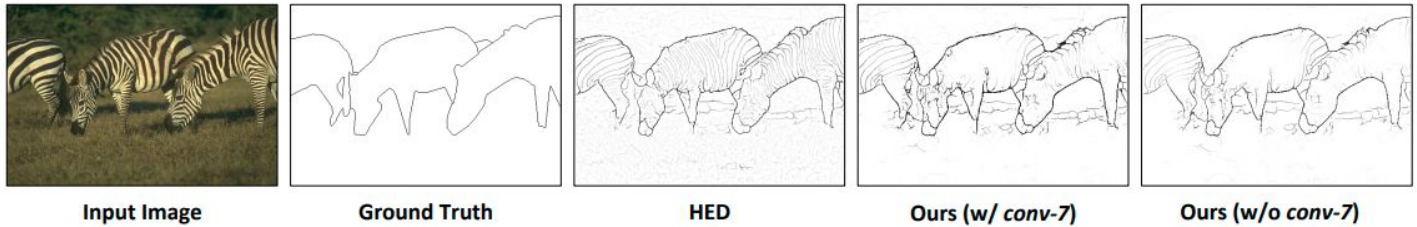


Figure 5. Qualitative results for edge detection. Notice that our approach generates more semantic edges for *zebra* compared to HED [94]. There are more similar examples of *eagle*, and *giraffe* in Appendix C. Best viewed in the electronic version.

5.1. Semantic Segmentation

Training: For all the experiments we used the publicly available *Caffe* library [49]. All trained models and code will be released. We make use of ImageNet-pretrained values for all convolutional layers, but train our MLP layers "from scratch" with Gaussian initialization ($\sigma = 10^{-3}$) and dropout [85] ($r = 0.5$). We fix momentum 0.9 and weight decay 0.0005 throughout the fine-tuning process. We use the following update schedule (unless otherwise specified): we tune the network for 80 epochs with a fixed learning rate (10^{-3}), reducing the rate by $10\times$ twice every 8 epochs until we reach 10^{-5}.

Dataset: The PASCAL-Context dataset [4] augments the original sparse set of PASCAL VOC 2010 segmentation annotations [26] (defined for 20 categories) to pixel labels for the whole scene. While this requires more than 400 categories, we followed standard protocol and evaluate on the 59-class and 33-class subsets. The results for PASCAL VOC-2012 dataset [26] are in Appendix A.

Evaluation Metrics: We report results on the standard metrics of pixel accuracy (AC) and region intersection over union (IU) averaged over classes (higher is better). Both are calculated with DeepLab evaluation tools[3].

Results: Table 12 shows performance of our approach compared to previous work. Our approach without CRF does better than previous approaches based on it. Due to space constraints, we show only one example output in Figure 6 and compare against FCN-8s [62]. Notice that we capture fine-scale details, such as the leg of birds. More analysis and details are in Appendix A.

[3] https://bitbucket.org/deeplab/deeplab-public/

Model	59-class		33-class	
	AC (%)	IU (%)	AC (%)	IU (%)
FCN-8s [61]	46.5	35.1	67.6	53.5
FCN-8s [62]	50.7	37.8	-	-
DeepLab (v2 [15])	-	37.6	-	-
DeepLab (v2) + CRF [15]	-	39.6	-	-
CRF-RNN [101]	-	39.3	-	-
ConvPP-8 [93]	-	**41.0**	-	-
PixelNet	**51.5**	**41.4**	**69.5**	**56.9**

Table 7. **Evaluation on PASCAL-Context [4]:** Most recent approaches [15, 93, 101] except FCN-8s, use spatial context post-processing. We achieve results better than previous approaches without any CRF. CRF post-processing could be applied to any local unary classifier (including our method).

5.2. Surface Normal Estimation

We use NYU-v2 depth dataset, and same evaluation criteria as defined earlier in Section 4. We improve the state-of-the-art for surface normal estimation [6] using the analysis for general pixel-level optimization. While Bansal et al. [6] extracted hypercolumn features from $1 \times 1 \times 4096$ *conv-7* of VGG-16, we provided sufficient padding at *conv-6* to

Stage 4: Read again from the beginning

Method	IoU (V)	Mean	Median	RMSE	11.25°	22.5°	30°
All Pixels	44.4	25.6	19.9	32.5	29.1	54.9	66.8
Random 4% Pixels	44.6	25.7	20.1	32.5	28.9	54.7	66.7

Table 1. **Sampling:** We demonstrate that sampling few pixels for each mini-batch yields similar accuracy as using all pixels. The results are computed on models trained for 10 epochs and 10,000 iterations for semantic segmentation and surface normal estimation, respectively.

Method	IoU (T)	Mean	Median	RMSE	11.25°	22.5°	30°
Linear (no bn)	3.6	24.8	19.4	31.2	28.7	56.4	68.8
Linear (bn)	62.4	22.5	16.1	29.7	37.0	62.8	73.3
MLP	**67.4**	**19.8**	**12.0**	**28.0**	**47.9**	**70.0**	**77.8**

Table 2. **Linear vs. MLP:** A multi-layer perceptron over hypercolumn features gives better performance over linear model without requiring normalization/scaling. Note: bn stands for batch-normalization.

4.2. Linear vs. MLP

Most previous approaches have focussed on linear predictors combining the information from different convolutional layers (also called 'skip-connections'). Here we con-

Model	#features	sample (#)	Memory (MB)	Disk Space (MB)	BPS
FCN-32s [62]	4,096	50,176	2,010	518	20.0
FCN-8s [62]	4,864	50,176	2,056	570	19.5
FCN/Deconvolution					
Linear	1,056	50,176	2,267	1,150	6.5
MLP	1,056	50,176	3,914	1,232	1.4
FCN/Sampling					
Linear	1,056	2,000	2,092	1,150	5.5
MLP	1,056	2,000	2,234	1,232	5.1
PixelNet/On-demand					
Linear	1,056	2,000	322	60	43.3
MLP	1,056	2,000	465	144	24.5
MLP (+*conv-7*)	5,152	2,000	1,024	686	8.8

Table 3. **Computational Requirements:** We record the number of dimensions for hypercolumn features from conv-$\{1_2, 3_3, 5_3\}$, number of samples (for our model), memory usage, model size on disk, number of mini-batch updates per second (BPS measured by forward/backward passes). We use a single 224×224 image as the input. We compared our network with FCN [62] where a deconvolution layer is used to upsample the result in various settings. Besides FCN-8s and FCN-32s here we first compute the upsampled feature map, and then apply the classifiers for FCN [62]. Clearly from the table, our approach require less computational resources as compared to other settings.

Quick Recap

- Use **Google scholar** to find papers.

Quick Recap

- Use **Google scholar** to find papers.
- **Skimming** papers - Abstract, Introduction, Figures, Conclusion

Quick Recap

- Use **Google scholar** to find papers.
- **Skimming** papers - Abstract, Introduction, Figures, Conclusion
- **AI tools** are helpful but we encourage discussions :)

Quick Recap

- Use **Google scholar** to find papers.
- **Skimming** papers - Abstract, Introduction, Figures, Conclusion
- **AI tools** are helpful but we encourage discussions :)
- **Skip math** equations or most technical details when reading the paper for the first time, you can always come back to it later.

Quick Recap

- Use **Google scholar** to find papers.
- **Skimming** papers - Abstract, Introduction, Figures, Conclusion
- **AI tools** are helpful but we encourage discussions :)
- **Skip math** equations or most technical details when reading the paper for the first time, you can always come back to it later.
- **Figures** are your best friend. Look for website link on the first page.

Quick Recap

- Use **Google scholar** to find papers.
- **Skimming** papers - Abstract, Introduction, Figures, Conclusion
- **AI tools** are helpful but we encourage discussions :)
- **Skip math** equations or most technical details when reading the paper for the first time, you can always come back to it later.
- **Figures** are your best friend. Look for website link on the first page.
- Keep in mind your **purpose** for reading and understanding the paper.

Try this method on your next paper and see what changes.

Please reach out to me if you have any doubts on this recitation at
ishitag@cs.cmu.edu