

HOMWORK 3

STRUCTURED SVM AND MAP INFERENCE¹

10-418 / 10-618 MACHINE LEARNING FOR STRUCTURED DATA (FALL 2019)

<https://piiazza.com/cmu/fall2019/1041810618>

OUT: Oct. 22, 2019

DUE: Nov. 06, 2019 11:59 PM

TAs: Aakanksha, Austin, Karthika

START HERE: Instructions

Summary In this assignment, you will implement a convolutional neural network (CNN) with a structured SVM for semantic segmentation. Section 1 will help you develop a better understanding of structured SVM and loss-augmented inference through some warm-up problems. Then, in Section 2, you will build on these intuitions to implement the CNN-SVM model and compare its performance with a vanilla CNN.

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., “Jane explained to me what is asked in Question 2.1”). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section on the course site for more information: <http://www.cs.cmu.edu/~mgormley/courses/10418/about.html#7-academic-integrity-policies>
- **Late Submission Policy:** See the late submission policy here: <http://www.cs.cmu.edu/~mgormley/courses/10418/about.html#6-general-policies>
- **Autolab:** You will submit your code for programming questions on the homework to Autolab (<https://autolab.andrew.cmu.edu/>). After uploading your code, we will manually grade your code by hand. We will not use Autolab to autograde your code.
- **Submitting your work to Gradescope:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, we will be using Gradescope (<https://gradescope.com/>). Please use the provided template. Submissions can be handwritten, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Regrade requests can be made, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. For short answer questions you **should not** include your work in your solution. If you include your work in your solutions, your assignment may not be graded correctly by our AI assisted grader.

For multiple choice or select all that apply questions, shade in the box or circle in the template document corresponding to the correct answer(s) for each of the questions. For \LaTeX users, replace `\choice` with `\CorrectChoice` to obtain a shaded box/circle, and don't change anything else.

¹Compiled on Friday 8th November, 2019 at 03:05

1 Written Questions [52 pts]

Answer the following questions in the template provided. Then upload your solutions to Gradescope. You may use \LaTeX or print the template and hand-write your answers then scan it in. Failure to use the template may result in a penalty. There are 52 points and 12 questions.

1.1 Semantic Segmentation

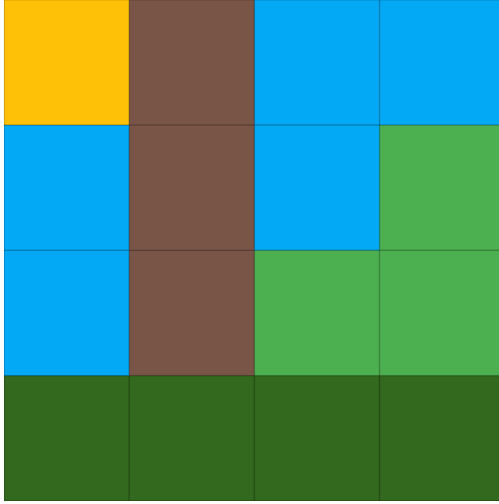


Figure 1.1: Input Image

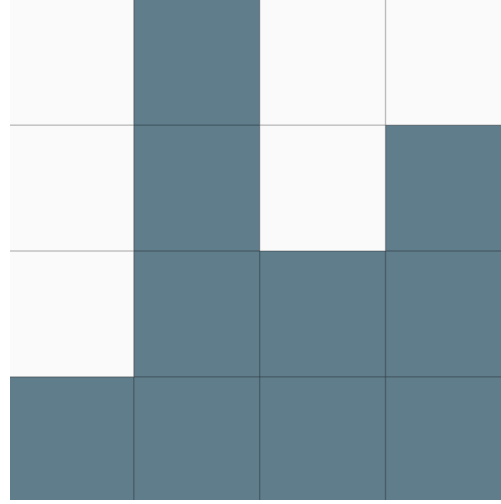


Figure 1.2: Image Segmentation

Let's assume we have a 4 x 4 grid consisting of RGB pixels. Our task is to perform image segmentation, i.e. assign a label to each pixel from the set $\{\text{foreground}, \text{background}\}$. Intuitively, neighboring pixels should have similar values in y , i.e. pixels associated with a mountain should form one continuous blob. This knowledge can be naturally modeled via a Potts model which consists of potentials $\phi(y_i, x)$ that encode the likelihood that any given pixel is from our subject and pairwise potentials $\phi(y_i, y_j)$ which will encourage adjacent y 's to have the same value with high probability. If visualized as a graphical model, we would have a node for each pixel and an edge for each pair of pixels that are touching vertically or horizontally but not diagonally. We will refer to these sets as V and E respectively.

As seen in class, solving this task corresponds to finding y such that

$$\begin{aligned} \hat{y} &= \operatorname{argmax}_y \log_w p(y|x) \\ &= \operatorname{argmax}_y \sum_j w^T \phi(x_j, y_j) + \sum_{j,k \in E} w^T \phi(y_j, y_k) \end{aligned}$$

NOTE: This is not the same model used in the programming section. We will provide the set-up for that separately.

1.2 MAP Inference with Integer Linear Programming

In this section, we will formulate our problem as an Integer Linear Program (ILP) and use it to predict the segmentation of a small patch of our image. For convenience, denote the background class with 0 and the foreground class with 1. Furthermore suppose we have variables α and β that we will be optimizing over. Let $\alpha_i(y)$ represent an indicator function that is one when pixel i is labeled class y . Let $\beta_{ij}(y_n, y_m)$ represent an indicator function that is one when pixel i is set to class y_n and pixel j is set to class y_m .

First we will need to translate our requirements into a constraint set.

1. (2 points) **Short answer:** Write constraints on the variables $\alpha_i(y)$ such that each pixel i is assigned one and only one class y .

$$\sum_{y \in \{0,1\}} \alpha_i(y) = 1 \quad \forall i$$

2. (4 points) **Short answer:** Write constraints on the variables $\beta_{ij}(y_n, y_m)$ such that pixels i and j are consistently applied to the same classes y_n and y_m in the pairwise potentials. Note that this will only be defined over ij pairs in our edge set E .

$$\sum_{y_m \in \{0,1\}} \beta_{ij}(y_n, y_m) = \alpha_i(y_n) \quad \forall (i, j) \in E$$

3. (2 points) **Short answer:** Using the score functions and goal defined above, write an ILP that represents our constraint set and objective such that if solved we would have a solution to our image segmentation task.

Maximize $\sum_j w^T \phi(x_j, y_j) + \sum_{j,k \in E} w^T \phi(y_j, y_k)$
s.t.

$$\begin{aligned} \sum_{y \in \{0,1\}} \alpha_i(y) &= 1 \quad \forall i \\ \sum_{y_m \in \{0,1\}} \beta_{ij}(y_n, y_m) &= \alpha_i(y_n) \quad \forall (i, j) \in E \\ \alpha_i(y) &\in \{0, 1\} \quad \forall \\ \beta_{ij}(y_n, y_m) &\in \{0, 1\} \end{aligned}$$

Suppose we had the following unary feature functions:

$$\phi_1(\text{yellow pixel, background}) = 1$$

$$\phi_2(\text{yellow pixel, foreground}) = 2$$

$$\phi_3(\text{brown pixel, foreground}) = 3$$

$$\phi_4(\text{brown pixel, background}) = 0$$

In addition suppose we had the following non-zero pairwise feature functions:

$$\phi_5(\text{background, background}) = 1$$

$$\phi_6(\text{foreground, background}) = 2$$

$$\phi_7(\text{background, foreground}) = 0$$

$$\phi_8(\text{foreground, foreground}) = 1$$

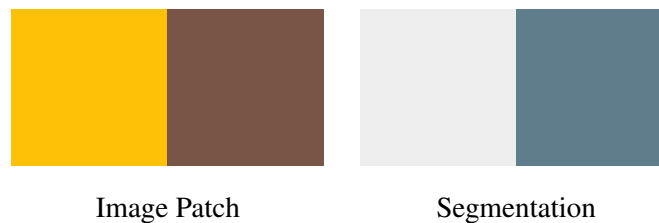


Figure 1.3: Extracted Image Patch and Segmentation

4. (7 points) **Short answer:** In this question we will walk through the process of solving the ILP for the 1×2 patch in the upper left hand corner of our image shown in figure 1.3. Assume an initial weight vector $w = [1, 1, 1, \dots]$.

(a) What are the scores assigned by the model for all possible y configurations shown below?

| | Yellow Background | Yellow Foreground |
|------------------|-------------------|-------------------|
| Brown Background | 2 | 4 |
| Brown Foreground | 4 | 6 |

(b) What are the settings of variable α that solves the above ILP?

We would label assign:

$$\alpha_0(\text{foreground}) = 1, \alpha_0(\text{background}) = 0$$

$$\alpha_1(\text{foreground}) = 1, \alpha_1(\text{background}) = 0$$

(c) What would be the updated weight vector w after running one iteration of structured perceptron?

$[0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 0 \ 2]$

5. (5 points) **Short answer:** Assume the same set-up as seen in question 4. Also assume an initial weight vector $w = [1, 1, 1, \dots, 1]$ and the Hamming loss as l .

(a) What are the loss augmented scores for all possible y configurations shown below?

| | Yellow Background | Yellow Foreground |
|------------------|-------------------|-------------------|
| Brown Background | 3 | 6 |
| Brown Foreground | 4 | 7 |

(b) What would be the updated weight vector w after running one iteration of structured SVM?

$[0 \ 2 \ 1 \ 1 \ 1 \ 1 \ 0 \ 2]$

1.3 Loss-Augmented Inference and Learning

1. (1 point) **Select all that apply:** Structured Perceptron differs from Structured SVM in?

■ Training

■ Inference

□ Evaluation

2. (4 points) Suppose you are given an MRF's factor graph $\mathcal{G} = (\mathcal{C}, \psi, \mathbf{y})$ consisting of a set of factors \mathcal{C} , potential functions ψ , and variables $\mathbf{y} \in \mathcal{Y}$, where \mathcal{Y} is the output space. Each factor $c \in \mathcal{C}$ touches a subset of the variables $\mathbf{y}_c = [y_{c_1}, y_{c_2}, \dots, y_{c_m}]^T$. The probability distribution defined by this graph decomposes multiplicatively according to the factors:

$$p_{\mathcal{G}}(\mathbf{y}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{y}_c)$$

Further, you are given a loss function that decomposes additively according to the factors:

$$\ell(\mathbf{y}^*, \hat{\mathbf{y}}) = \sum_{c \in \mathcal{C}} \ell_c(\mathbf{y}_c^*, \hat{\mathbf{y}}_c)$$

Show that you can define a new factor graph \mathcal{G}' (with the same factors and variables as \mathcal{G} , but different potential functions ψ') such that the most probable assignment to $\hat{\mathbf{y}}' \triangleq \arg\max_{\mathbf{y} \in \mathcal{Y}} p_{\mathcal{G}'}(\mathbf{y})$ solves the loss-augmented MAP inference problem for \mathcal{G} with loss ℓ .

Let $\psi'_c(\mathbf{y}_c) = \frac{\psi_c(\mathbf{y}_c)}{e^{\ell(\mathbf{y}_c^*, \mathbf{y}_c)}}$

Now we can see that

$$\begin{aligned} \hat{\mathbf{y}}' &\triangleq \arg\max_{\mathbf{y} \in \mathcal{Y}} p_{\mathcal{G}'}(\mathbf{y}) \\ &= \arg\max_{\mathbf{y} \in \mathcal{Y}} \sum_{c \in \mathcal{C}} \log \psi'_c(\mathbf{y}_c) \\ &= \arg\max_{\mathbf{y} \in \mathcal{Y}} \sum_{c \in \mathcal{C}} \log \psi_c(\mathbf{y}_c) - \ell(\mathbf{y}_c^*, \mathbf{y}_c) \end{aligned}$$

So we will be solving the loss-augmented MAP inference problem in the new factor graph for loss ℓ .

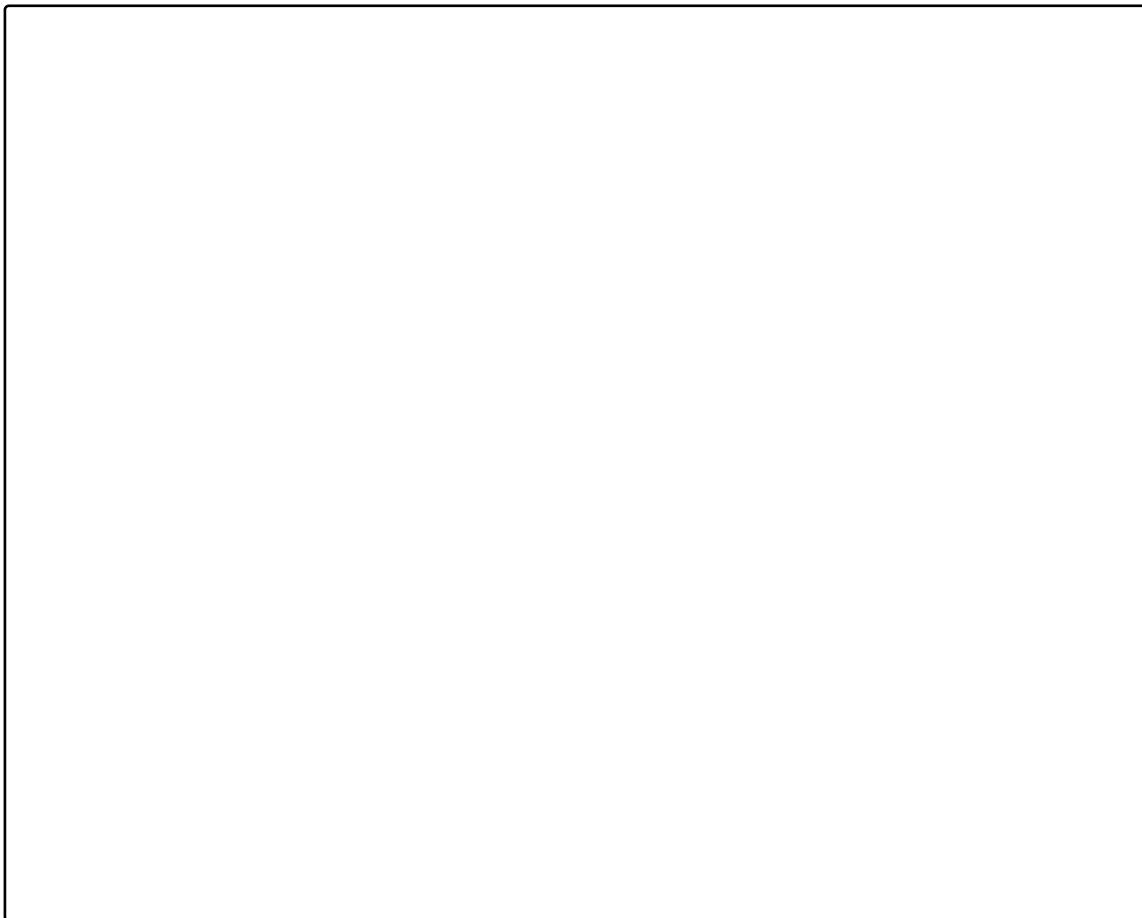
1.4 Empirical Questions

The following questions should be completed after you work through the programming portion of this assignment (Section 2).

1. (12 points) Report the mean IOU and pixel accuracy scores for all models. For this question, you should run the FCN for 2 epochs and the linear SVM and structured SVM for 3 epochs each. *Round each value to two significant figures*

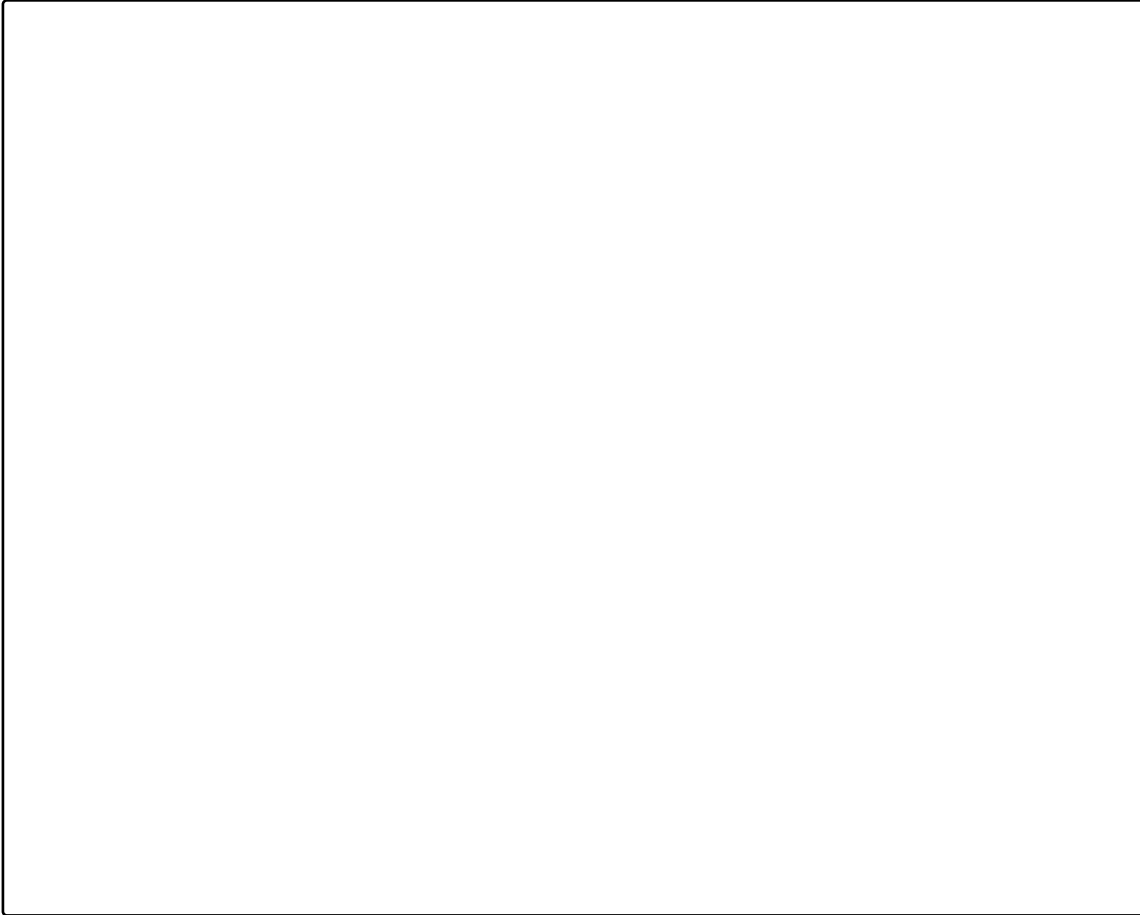
| Model | Mean IOU | Pixel Accuracy |
|-------------------|----------|----------------|
| FCN (Train) | | |
| FCN (Test) | 0.44 | 0.67 |
| FCN+l-SVM (Train) | | |
| FCN+l-SVM (Test) | 0.40 | 0.61 |
| FCN+s-SVM (Train) | | |
| FCN+s-SVM (Test) | 0.41 | 0.75 |

2. (10 points) Plot training and testing curves for the hinge loss (FCN+l-SVM) and structured hinge loss (FCN+s-SVM) respectively over 3 epochs. *Note: Your plot must be machine generated.*



3. (3 points) For any image of your choice from the test set, plot the labels predicted by FCN, FCN+l-

SVM and FCN+s-SVM as grayscale images. Use the visualization function provided in the code to plot images. Do you see any remarkable differences between the model predictions?



1.5 Wrap-up Questions

1. (1 point) **Multiple Choice:** Did you correctly submit your code to Autolab?

☐ Yes

☐ No

2. (1 point) **Numerical answer:** How many hours did you spend on this assignment?.

1.6 Collaboration Policy

After you have completed all other components of this assignment, report your answers to the collaboration policy questions detailed in the Academic Integrity Policies for this course.

1. Did you receive any help whatsoever from anyone in solving this assignment? If so, include full details including names of people who helped you and the exact nature of help you received.

2. Did you give any help whatsoever to anyone in solving this assignment? If so, include full details including names of people you helped and the exact nature of help you offered.

3. Did you find or come across code that implements any part of this assignment? If so, include full details including the source of the code and how you used it in the assignment.

2 Programming [48 pts]

Your goal in this assignment is to implement a structured SVM on top of a convolutional neural network for image segmentation. Given an image as a 32x32 grid of pixels, you will assign a label from the set $\{foreground, background\}$ to each pixel.

Your solution for this section must be implemented with **PyTorch** and **Google OR-Tools**² using the data files we have provided to you. This restriction is because we will be grading your code by hand to check your understanding as well as your model's performance.

2.1 Task Background

Semantic segmentation is the task of assigning a class label to each pixel in an image. This task can be thought of as image classification, but at the pixel level. Figure 2.1 shows a sample image with its corresponding segmented image.



Figure 2.1: Sample image from the PASCAL VOC-2012 Image Segmentation dataset

Semantic segmentation is useful for a variety of applications such as autonomous vehicles, photo-editing tools and many tasks in robotics. In this question, we will first train a simple semantic segmentation model using a fully convolutional network (FCN). Next, we will use this FCN as a feature extractor to generate 100-dimensional features for each pixel in the image. Using these pixel features, we will build two semantic segmentation models. The first baseline model is a simple linear support vector machine (SVM) which uses the FCN features and predicts a class label for each pixel in the image independently. The second model is a structured SVM which takes in FCN features and predicts class labels for all pixels in the image while incorporating interactions between neighboring pixels. Following subsections provide more details about the dataset, metrics and model structure.

2.2 Data

For this task, we will be using the semantic segmentation dataset from the PASCAL VOC 2012 Challenge.³ This dataset consists of 2913 examples in total (2330 for training and 583 for testing). For each example in this dataset, we have a pair of images: the original image and the segmented image. To make it easier to work with the data, we have converted segmented images into numpy arrays of class labels. To ensure reasonable runtime, we have also cropped and downsampled all images to 32x32 pixels. Finally, instead of having multiple class labels (eg: person, aeroplane etc.), we will treat this as a binary classification task with the label set $\{foreground, background\}$.

2.3 Metrics

To evaluate model performance on this dataset, we will be using two metrics:

²<https://developers.google.com/optimization>

³<http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html>

- **Pixel Accuracy:** Pixel-level classification accuracy for each image computed as

$$\frac{\#CorrectPixels}{\#TotalPixels} \quad (2.1)$$

- **Mean IOU (Intersection-over-Union):** Mean of per-class IOU (intersection-over-union) across all classes for each image. For one class (eg: foreground), IOU is computed as

$$\frac{\#CorrectForegroundPixels}{\#GoldForegroundPixels + \#PredictedForegroundPixels - \#CorrectForegroundPixels} \quad (2.2)$$

We will average these metrics across all images

2.4 Fully Convolutional Network

As a first step towards building a semantic segmentation model, we will train a fully convolutional network. A fully convolutional network differs from a regular convolutional neural network in the later half of its architecture. While a convolutional neural network returns a dense representation for each image (eg: a 6x6 representation for a 32x32 image), a fully convolutional network returns a representation of the same size as the original image. FCNs use regular convolution layers to first compress an image into a dense representation before *deconvolving* the dense representation to get a tensor of the same size as the original image (as shown in figure 2.2).

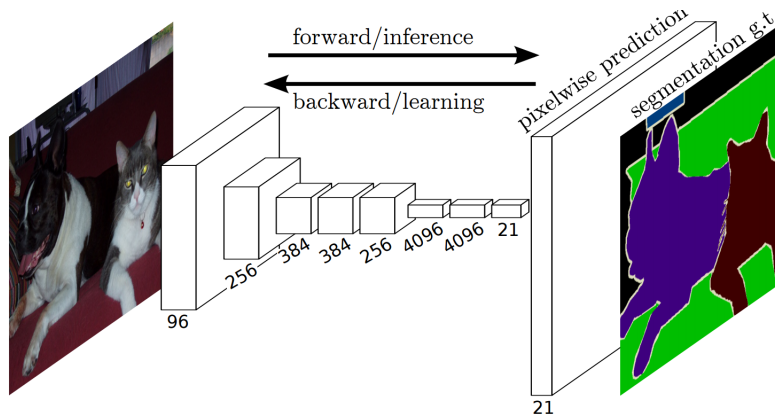


Figure 2.2: Fully convolutional network for semantic segmentation

For this assignment, we have provided you with code to train and test a FCN. Our FCN implementation uses the first 5 layers from AlexNet as convolution layers and an interpolation layer to perform the deconvolution. After running the FCN on a 32x32 image, you should be able to extract a 32x32x100 tensor which contains a 100-dimensional feature vector for each pixel. We add an additional 100x2 linear layer to predict (foreground, background) scores for each pixel and train this network using cross-entropy loss. After training, we discard the linear layer and use the FCN weights to generate pixel-level features for a given image.

2.5 Baseline

The first model you will be implementing is a simple linear support vector machine. This model takes as input a 32x32x100 tensor computed by the FCN and uses weights w_{fc} to calculate per-class scores for each

pixel independently. This is analogous to simply adding a linear layer on top of the FCN. However, since the baseline is a linear SVM, it is trained using hinge loss. (**Hint:** To implement hinge loss in PyTorch, you might find it helpful to look up `MultiMarginLoss()`).

2.6 Main Model

The second model you will be implementing is a structured support vector machine. Unlike the linear SVM, this model does not predict labels for each pixel independently. This is useful because adjacent pixels tend to belong to the same class, indicating that there may be some benefits to treating this problem as a structured prediction problem and leveraging pixel interactions. In this assignment, we will model the structure of the output using a formulation based on associative markov networks.⁴ Associative markov networks are a class of undirected graphical models where each node is a discrete variable and clique potentials tend to favor the same label for all nodes in the clique. In our task, we treat each pixel as a node and only model pairwise interactions between pixels (i.e no cliques larger than 2 exist). For tractability, we further restrict pairwise interactions to non-diagonal neighboring pixels (up, down, left, right). This results in a lattice structure as shown in the figure:

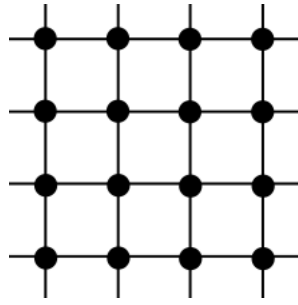


Figure 2.3: Output structure used by structured SVM

Given an input image as a $32 \times 32 \times 100$ feature tensor T extracted from the FCN, a SVM over this structured output space can be formulated as follows:

- **Variables:** $x_{i,c}$ where $i \in [0, 32 * 32)$, $c \in \{0, 1\}$, $y_{i,j,c}$, where $i \in [0, 32 * 32)$, $j \in [0, 32 * 32)$, $c \in \{0, 1\}$. $x_{i,c}$ and $y_{i,j,c}$ are pixel assignment and edge assignment variables respectively. $x_{i,c} = 1$ indicates that pixel at position i belongs to class c , while $y_{i,j,c} = 1$ indicates that pixels in neighboring positions i and j both belong to class c and are linked to one another.
- **Weights:** $w_{f,c}$ and $w_{f',c}$, where $f \in [0, 100)$, $f' \in [0, 200)$, $c \in \{0, 1\}$. $w_{f,c}$ and $w_{f',c}$ are pixel weights and edge weights respectively. These weights can be used to compute unary pixel potentials and edge potentials as $\phi_{i,c} = w_{f,c}T(i)$ and $\phi_{i,j,c} = w_{f',c}\text{concat}(T(i), T(j))$. Recall that we used a similar process in homework 2 to compute unary/ edge potentials for nodes in the parse tree from LSTM hidden states.
- **ILP formulation for MAP inference:** Using the pixel and edge potentials computed using weights and FCN features, the highest scoring output structure (i.e. the assignment of values to $x_{i,c}, y_{i,j,c}$) is chosen by maximizing the following objective:

$$\sum_i \sum_c \phi_{i,c} x_{i,c} + \sum_i \sum_j \sum_c \phi_{i,j,c} y_{i,j,c}$$

⁴<https://ai.stanford.edu/~koller/Papers/Taskar+al:ICML04.pdf>

subject to the following constraints:

$$\begin{aligned}\sum_c x_{i,c} &= 1 \\ y_{i,j,c} &\leq x_{i,c} \\ y_{i,j,c} &\leq x_{j,c}\end{aligned}$$

The first constraint ensures that each pixel is assigned to exactly one class, while the next two constraints ensure that if an edge exists between two pixels, they both belong to the same class. Assume that the highest-scoring structure after solving this optimization is denoted as \hat{O} while the correct structure is O^*

- **Loss function:** The structured hinge loss for this SVM is as follows:

$$L = \max(0, S(\hat{O}, T) + \ell(\hat{O}, O^*) - S(O^*, T))$$

where $\ell(\hat{O}, O^*)$ is the Hamming loss between the two outputs and the scoring function $S(O, T)$ is the same as the ILP objective (note that the potentials ϕ in the objective are computed using vectors from T).

Putting everything together, given an image I , you should run the following procedure:

1. Extract feature tensor $T = FCN(I)$
2. Compute pixel and edge potentials $\phi_{i,c}, \phi_{i,j,c}$ using T
3. Perform MAP inference via ILP to get highest scoring structure \hat{O}
4. Compute structured hinge loss with \hat{O}
5. Backpropagate through the structured loss to update weights

At test time, you only need to perform steps 1-3 and use the highest scoring assignment as your output.

2.7 Implementation Details

We have provided you with some starter code (`starter_code.py`). You should be able to simply run this starter code to train and test a FCN on the data after adding the correct paths. You are expected to implement both SVM models by filling in the TODOs. The reference SVM implementation uses Adam optimizer with learning rate 0.0001 for both linear and structured SVM. It also uses class weights for the linear model (similar to the FCN) to handle the class imbalance issue. You should train the FCN in isolation for 2 epochs before using it as a feature extractor for any of the SVM models. Both SVM models only update their own weights and DO NOT update the FCN weights. Train both SVM models for 3 epochs each before answering the empirical questions. For ILP optimization, use the GLOP linear solver from OR-Tools. Though this solver optimizes the LP-relaxation instead of the ILP, the associative markov networks paper demonstrates that for our objective and constraints, the LP-relaxation and ILP have the same solution. (However, if the number of classes is greater than 2, this result does not hold!)

2.8 Autolab Submission [35 pts]

You must submit a .tar file named `structuredsvm.tar` containing `structuredsvm.py`, which contains all of your code.

You can create that file by running:

```
tar -cvf structuredsvm.tar structuredsvm.py
```

from the directory containing your code.

Some additional tips: **DO NOT** compress your files; you are just creating a tarball. Do not use `tar -czvf`. **DO NOT** put the above files in a folder and then tar the folder. Autolab is case sensitive, so observe that all your files should be named in **lowercase**. You must submit this file to the corresponding homework link on Autolab.

Your code will **not** be autograded on Autolab. Instead, we will grade your code by hand; that is, we will read through your code in order to grade it. As such, please carefully identify major sections of the code via comments.