CSE 151B Deep Learning
Assignment 1 Part 2 + Assignment 2 Part 1

Summer Session 1 2026

Instructions

(A1 Part 2) All questions in "Understanding Convolutional Network Basics" and questions 1 and 2 in "Fully Convolutional Network for Semantic Segmentation" is encouraged to be finished by Wednesday, July 8, 2026 (11:59 PM) and is due Wednesday, July 15, 2026 (11:59 PM)

(A2 Part 1) "Fully Convolutional Network for Semantic Segmentation" questions 3 through 5 due on Wednesday, July 15, 2026 (11:59 PM)

1. For completing "Fully Convolutional Network for Semantic Segmentation" questions 1 and 2 as a part of A1, please upload only your **util.py and pixel.txt files**.
2. For completing "Fully Convolutional Network for Semantic Segmentation" questions 3 through 5 as a part of A2, please submit **all of the source code files, trained models, and a *README.md* file** that includes detailed instructions on how to run your code.

   You should write clean code with consistent format, as well as explanatory comments. Do not submit any of your output plot files or .pyc files, just the .py files and a README.md that explains how to run your code.
3. Any form of copying, plagiarizing, grabbing code from the web, having someone else write your code for you, etc., is cheating. We expect you all to do your own work, and when you are on a team, to pull your weight. Team members who do not contribute will not receive the same scores as those who do. Discussions of course materials and homework solutions are encouraged, but you should work on the final solutions to the architectures & algorithms part alone. Books, notes, and Internet resources can be consulted, but not copied from. Working together on homework must follow the spirit of the **Gilligan's Island Rule** (Dymond, 1986): No notes can be made (or recording of any kind) during a discussion, and you must watch one hour of Gilligan's Island or something equally insipid before writing anything down. Suspected cheating has been and will be reported to the UCSD Academic Integrity office.

   In addition, as stated in the syllabus, you may use chatgpt or copilot, but you must indicate in the text or code where you used it, and include an appendix with the prompt you used, the output of the AI model, and how you modified it, if you did. Furthermore, you must understand your code, so that you can explain it in a code review.
4. **Start early!** If you have any questions or uncertainties about the assignment instructions, please ask about them as soon as possible (preferably on Piazza, so everybody can benefit). We want to minimize any possible confusion about this assignment and have tried very hard to make it understandable and easy for you to follow.
5. You will be using PyTorch (v 2.1), a Deep Learning library, for the tasks in this assignment. Also you are allowed to only use Numpy, Matplotlib and PIL Imaging library apart from the libraries that come along with the Python installation. Steps to access the UCSD GPU cluster are explained in the following sections.
6. This PA requires the use of a GPU. UCSD provides students with a GPU cloud instance via the UCSD Datahub. More information about it at the end of this PA. Remember to start early as there are a limited

number of GPUs available and they *will* get impacted as the deadline approaches. (Feel free to use your own GPU enabled systems if you want to!)

Learning Objectives

1. Understand the basics of convolutional neural networks, including convolutional and de-convolutional layer mechanics, max-pooling, and dimensions of layers.
2. Learn how to implement a CNN architecture in PyTorch for Semantic Segmentation using best practices.
3. Build intuition on the effects of modulating the design of a CNN by experimenting with your own (or 'classic') architectures.
4. Learn best practices of transfer learning model by fine tuning a model for Semantic Segmentation.

Understanding Convolutional Network Basics

This portion of the assignment is to build your intuition and understanding of the basics of convolutional networks - namely how convolutional layers learn feature maps and how max pooling works - by manually simulating these. We expect each team member to complete this task individually.

For questions 1-4, consider the $(5 \times 5 \times 2)$ input volume with values in range $[-1, 1]$ and the corresponding $(3 \times 3 \times 2)$ filter shown below. The terminology here follows that in the Stanford convnet course page. On that page search down for "Convolution Demo". This example corresponds to the one given there, except that there are only 2 input channels (5x5x2), and one (3x3x2) filter. For Question 1, the filter should be applied to the channels next to them just like on the Stanford page. You can assume a bias of zero.

1 Questions

Input Volume (5x5x2)

0	1	1	1	1
1	1	0	-1	0
1	0	-1	-1	1
0	-1	-1	0	1
1	0	-1	1	0

1	1	1	0	0
0	-1	1	1	0
0	1	-1	-1	1
-1	0	-1	-1	1
1	1	-1	0	0

Filter (3x3x2)

1	1	0
1	0	-1
0	-1	-1

0	1	1
-1	0	1
-1	-1	0

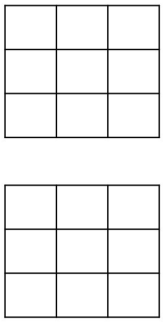
1. In the provided table below, fill in the values resulting from applying the above filter to the above input volume with no zero-padding and a stride of 1. Don't apply any activation function - this is just the weighted sum of the inputs (what we've been calling a). **There will be unused cells** - place an $\times$ in them. (3 pts)

Output Feature Map

2. What kind of features do the kernels in Part 1 detect? (just describe them in English) (1 pt)

3. What is the "ideal" $3 \times 3 \times 2$ input volume that would maximally activate the $3 \times 3 \times 2$ filter? Recall that the input "pixels" can only be drawn from the set {-1,0,1}. Fill in these maximally activating values in the area below. If there are values that would not make any difference, place an $\times$ in them. (2pts)

Maximally
Activating Patch



4. **Spatial pooling:** Using your **output feature map** in question 1, apply *max-pooling* using a $[2 \times 2]$ kernel with a stride of 1. Recall from lecture that spatial pooling gives a CNN invariance to small transformations in the input, and max-pooling is more widely used over sum or average pooling due to empirically better performance. (2 pts)

Max Pooled
Output

5. **Dilated Convolutions:** Maxpooling helps the network learn the global context of input feature maps but loses spatial resolution. Dilated convolutions allow for larger filters without adding additional parameters. You can read more on Dilated convolutions here. Consider the same input feature map used for question 1. Also consider the same filters provided for question 1. Fill in the value below after dilated convolution (with dilation factor 2) between the input volume and the filter. (2 pts)

Output Feature
Map

6. **Number of learnable parameters and feature map dimensions**
Reference to Stanford convnet course page and PyTorch conv2d documentation page for this question. Suppose we had two architectures:

```
inputs -> conv1 -> conv2 -> maxpool1 -> fc1 (outputs)
```

```
inputs -> conv3 -> maxpool2 -> fc2 (outputs)
```

- All convolutions have stride 1 and no zero padding unless mentioned. conv1 has a 3x3 kernel with 3 output channels. conv2 has a 3x3 kernel with 3 output channels. maxpool1 has 3x3 kernel with stride 2. fc1 is a fully-connected layer between maxpool1 and 10 class outputs.

- conv3 has 3 output channels with a 5x5 kernel. maxpool2 has 3x3 kernel with stride 2. fc2 is a fully-connected layer between maxpool2 and 10 class outputs.
- If ReLU is the activation function between each layer and the inputs are [32 × 32] RGB images, answer following questions: (answering feature maps using this format: `NCHW` - batch size, number of channels, height, width)

(a) What are the filter bank shapes (in channel, out channel, height, width) of **conv1**, **conv2** and **conv3**?

(b) What is the output feature map shape of **conv1** and **conv2**?

(c) What is the output feature map shape after **conv3** is applied?

(d) What are the feature map shapes after **maxpool1** and **maxpool2** are applied?

(e) What are the number of weights in **fc1** and **fc2**? Don't forget to include the bias!

(f) What is the total number of parameters for the two networks? Don't forget to include the bias!

Show your work with calculations if necessary (6 pts)

Fully Convolutional Network for Semantic Segmentation

Problem statement: Deep convolutional neural networks have been applied to a broad range of problems and tasks within Computer Vision. In this part of the assignment, we will explore *semantic segmentation* task. Semantic segmentation means that every pixel in an image is classified as belonging to some category. This means that at the output level, there is a softmax over all of the categories for *each pixel in the image*. This is very important in the self-driving car field. We will build deep CNN architectures and train them from scratch to predict the segmentation masks of the images. (As mentioned in the instructions, for training we would require the use of GPU enabled hardware. More information available in the Datahub section at the bottom)

We will be using **PASCAL VOC-2012** dataset for the task of pixel-level semantic segmentation. This data set has pixel-wise annotation for 20 object categories (21 including the background category). More details about the dataset can be found here. The main goal of this challenge is to recognize objects from a number of visual object classes in realistic scenes (i.e., not pre-segmented objects). In this assignment, we will be using 21 categories. It is fundamentally a supervised learning learning problem in that a training set of segmented, dlabeled images is provided. Five examples, their labels, and overlapped images are shown below.

Implementation Instructions

Please familiarize yourself with Pytorch before implementing. One of the good places to start with Pytorch is this playlist.
We have provided you with a data loader: a custom PyTorch Dataset class-**voc.py**, specifically designed for this dataset. Please familiarize yourself with the code and read the comments.

Similar to part one of A1, please save your models and make them loadable using an "–experiment" flag.

The following are the implementation steps to follow.

1. **Evaluation metrics**

   Before we discuss the neural network details, we must clarify how you will accurately and transparently evaluate your model's performance. Here are some of the essential metrics for this:

   (a) Pixel Accuracy: percent correct predictions = $\frac{\text{total correct predictions}}{\text{total number of samples}}$ The issue with this measure is called *class imbalance*. When our classes are extremely imbalanced, it means that a class or some classes dominate the image, while some other classes make up only a small portion of the image. This means the network can reduce the error considerably just by labeling everything with the majority class. Unfortunately, class imbalance is prevalent in many real world datasets, so it can't be ignored. Therefore, there are alternative metrics that are better at dealing with this issue.

   (b) Intersection-Over-Union (IoU, Jaccard Index): Simply put, the IoU is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth. It is computed on a per-class basis, and then these are averaged over the classes. It is measured as, IoU = $\frac{TP}{TP+FP+FN}$, where TP, FP, and FN are the numbers of true positive, false positive, and false negative pixels, respectively, determined over the whole validation set. See this blog for an example of the difference between pixel accuracy and IoU. For a particular class, let's say car, TP is the number of pixels for which the ground truth is car and the network labels them as car. FP is the count of pixels that the network labeled as car, but weren't cars. FN is the number of pixels that the network should have labeled car, but didn't.

   Complete the implementation of these functions in the **utils.py** file provided. A detailed explanation of the class label ids and their mapping is provided in **voc.py**.

   For each experiment in questions 3 through 5, please report the following according to the print format provided in the starter code:

   (a) Test Set Average Pixel Accuracy; Consider the object boundary as part of background (make the pixels with value 255 in the target to zero (label for background))

   (b) Test Set Average IoU; Consider the object boundary as part of background (make the pixels with value 255 in the target to zero (label for background))

   The print statement should look like the following:
   "Final average pixel accuracy: xxxx, test set average IoU: xxxx"

   Here are some **estimated thresholds** for a high pass for different experiments in this assignment. Note that this may change (in your favor) depending on the overall class performance! IoU tracks pixel accuracy closely, so it is omitted from these thresholds.

   - baseline - Pixel accuracy: 0.76
   - improved_baseline - Pixel accuracy: 0.72
   - experimental - Pixel accuracy: 0.70

   (experimental varies a lot based on what architecture you use, pixel accuracy threshold is there just to catch completely incorrect submissions)

2. **Dataset loading**

   (a) Download the dataset using the starter code - **download.py** (Run on terminal using `python3 download.py` and make sure you are in the right directory. => it should create a data folder in root directory.

   (b) **voc.py** contains custom PyTorch Dataset class which is already completed for you. Here, we have also provide a necessary image transformation for this assignment: resizing every image to 224×224. However, you will need to write code for any other data pre-processing or transformations you want to use (such as normalization, cropping, horizontal flipping, etc.). For list of available transformation, have a look at PyTorch transformations here. (**Note:** For some transformations (such as cropping/flipping/mirroring), you have to make sure the same transformation is applied to both image and its label).

   (c) Dataloader in **train.py** uses the dataset object and iterates over your dataset while training.

   To answer: Please report the RGB value of the 7th pixel (index 6) on the 100th image (index 99) in the VOC test set in a text file called **"pixel.txt"** in the format "R: 0.xxxx, G: 0.xxxx, B: 0.xxxx". (Hint: the 100th test set image should have a person looking at the camera on a green background.)

3. **Baseline Model** An outline for creating the model is provided in **basic_fcn.py** for the purposes of (i) getting acquainted with PyTorch, (ii) getting initial results to compare with more complex architectures and approaches to solving this semantic segmentation problem. The baseline architecture contains an encoder and a decoder structure. Using the starter code provided, complete the implementation for the architecture as described in the comments, replacing the instances of __ with its corresponding missing value. This includes finishing the __init__() and forward() functions.

   To run the **basic_fcn.py** model, we have additionally provided a **train.py**, which you will have to complete to train your model. Based on your reasoning for determining the activation function for the output layer of the network, determine which loss criterion you should be optimizing - this may be something you are already familiar with (note that PyTorch loss criteria can be found in the **torch.nn** package). Additionally, use the Adam/AdamW gradient descent optimizer, which can be found in the **torch.optim** package. Use early stopping to train the baseline model. A naive implementation of this model should reach average pixel accuracy of 0.75 and average IoU of 0.09 approximately on the validation dataset. **Please store this baseline model in a folder named "models" and make it loadable with the flag –experiment 'baseline' in train.py.**

   Please note that this is a very bare-bones implementation which might not perform very well. As such, this architecture may have 'good' overall classification accuracy, but fail to address the class-imbalance problem.

4. **Improving baseline model:** The baseline model we trained above is a basic implementation which can be improved. Please choose one of the following improvements to implement. **Please store this improved model in a folder named "models" and make it loadable with the flag –experiment 'improved_baseline' in train.py.**

   (a) It is usually a good idea to adjust your learning rate with the number of epochs during training. Try using the cosine annealing learning rate scheduler and see if it helps improve the performance.

   (b) Augment your dataset by applying transformations to the input images: this can include using mirror flips, rotating the images slightly, or different crops. Make sure to apply the same transformations to the corresponding labels!

   (c) Address the rare class or imbalanced class problem. This is often done by pressuring the network to categorize the infrequently seen classes. You can use weighted loss, which weights infrequent classes more. You must implement your own weighted/balanced loss criterion.

5. **Experimentation and your solution:** To get a better sense of how your design choices affect model performance, we encourage you to experiment with various approaches to solving this semantic segmentation problem using a deep CNN. You are welcome to make a copy of the **basic_fcn.py** file and **train.py** for this purpose. Use all techniques to improve the model performance and handle the problem of class imbalance. To receive full credit, you are not expected to outperform the baseline. Please choose one of the following experiments to implement. **Please store this modified model in a folder named "models" and make it loadable with the flag –experiment 'experimental' in train.py.**

(a) Try and come up with an architecture of your own - this includes making significant changes in the number of layers, activation functions, dimensions of the convolutional filters, etc. Simply adding an additional layer is not sufficient! You can take inspiration from already exisiting models and try to combine their strengths.

(b) Try Transfer Learning with any of the architectures present in the Pytorch Library to replace the encoder part of the given FCN architecture. Does this improve the performance? Why do you think you observe the changes that you do? (NOTE: You might have to make necessary changes to use existing architecture. For example, if you want to use ResNet34 pretrained on Imagenet, you can remove the last fully connected layer and the avgpool layer before it to match the dimension for your decoder.)

(c) Refer to the U-Net architecture in this paper and implement the same architecture for the given dataset. Note that this architecture is very similar to the FCN architecture provided with some important changes. If you are interested enough, you can also try the data augmentation methods they mentioned in the paper.

Environment Set Up

UCSD GPU cluster

UCSD provides access to its GPU cluster through UCSD Datahub. You are free to use any other platforms but we won't be responsible for any glitches, bugs or delays in them.
(NOTE: The steps here might be outdated as the quarter proceeds. We advise you to check Piazza and their official website for any changes in the steps.)
Steps to access UCSD datahub are as follows.

1. **Method 1** (recommended)
   (a) Log into UCSD Datahub using Single Sign On method.
   (b) There will be two notebooks visible for the class. Choose the one that allocates you GPU.
   (c) Once the notebook is spawned, A JupyterHub Dashboard will appear.
2. **Method 2** (not verified)
   (a) ssh into dsmlp-login.ucsd.edu using *ssh username@dsmlp-login.ucsd.edu*. Before that, you will need to connect through VPN if you are not connected to on campus network.
   (b) Use *launch-scipy-ml-gpu.sh* to launch a Jupyter Notebook with GPU access.
   (c) More indepth detail available here.

More details on using Datahub is provided on this link

Local Environment

If you have access to local NVIDIA GPU, you can use the following instructions to set up environment for this PA only.

- In a conda environment, navigate to directory with file `environment.yml` (from starter code)
- Run the following line: `conda env create -f environment.yml`
- Activate the environment with `conda activate 151bpa2` (name inside `environment.yml`)
- If needed, launch the classic jupyter notebook with `jupyter nbclassic`

Start early on the PA

Datahub GPUs might get over utilized towards the end of PA and you may not be able to get access to GPU when you need it. Good Luck and Happy Coding !! :)

Acknowledgments

All parts of this assignment were directly adapted from Professor Gary Cottrell's Winter 2025 offering of the course.