EECS 182 Deep Neural Networks
Fall 2023 Anant Sahai

Homework 4

This homework is due on Sunday, September 24, 2023, at 10:59PM.

1. Desigining 2D Filter (Coding Question)

Convolutional layer, which is the most important building block of CNN, actively utilizes the concept of filters used in traditional image processing. Therefore, it is quite important to know and understand the types and operation of image filters.

Look at HandDesignFilters.ipynb. In this notebook, we will design two convolution filters by hand to understand the operation of convolution:

(a) **Blurring filter.**

(b) **Edge detection filter.**

For each type of filter, please **include the image generated by the Jupyter Notebook in your submission to the written assignment**. The image should be added to the PDF document that you will be submitting.

2. Feature Dimensions of Convolutional Neural Network

In this problem, we compute output feature shape of convolutional layers and pooling layers, which are building blocks of CNN. Let's assume that input feature shape is $W \times H \times C$, where W is the width, H is the height and C is the number of channels of input feature.

(a) A convolutional layer has 4 architectural hyperparameters: the filter size(K), the padding size (P), the stride step size (S) and the number of filters (F). **How many weights and biases are in this convolutional layer? And what is the shape of output feature that this convolutional layer produces?**

(b) A max pooling layer has 2 architectural hyperparameters: the stride step size(S) and the "filter size" (K). **What is the output feature shape that this pooling layer produces?**

(c) Let's assume that we have the CNN model which consists of L successive convolutional layers and the filter size is K and the stride step size is 1 for every convolutional layer. Then **what is the receptive field size of the last output**?

(d) Consider a downsampling layer (e.g. pooling layer and strided convolution layer). In this problem, we investigate pros and cons of downsampling layers. This layer reduces the output feature resolution and this implies that the output features lose a certain amount of spatial information. Therefore when we design CNNs, we usually increase tchannel length to compensate this loss. For example, if we apply a max pooling layer with a kernel size of 2 and a stride of 2, we increase the output feature size by a factor of 2. **If we apply this max pooling layer, how much does the receptive field increases? Explain the advantage of decreasing the output feature resolution with the perspective of reducing the amount of computation.**

(e) Let's take a real example. We are going to describe a convolutional neural net using the following pieces:

- CONV3-F denotes a convolutional layer with F different filters, each of size $3 \times 3 \times C$, where C is the depth (i.e. number of channels) of the activations from the previous layer. Padding is 1, and stride is 1.
- POOL2 denotes a 2×2 max-pooling layer with stride 2 (pad 0)
- FLATTEN just turns whatever shape input tensor into a one-dimensional array with the same values in it.
- FC-K denotes a fully-connected layer with K output neurons.

Note: All CONV3-F and FC-K layers have biases as well as weights. **Do not forget the biases when counting parameters.**

Now, we are going to use this network to do inference on a single input. **Fill in the missing entries in this table of the size of the activations at each layer, and the number of parameters at each layer. You can/should write your answer as a computation (e.g.** $128 \times 128 \times 3$**) in the style of the already filled-in entries of the table.**

Layer	Number of Parameters	Dimension of Activations
Input	0	$28 \times 28 \times 1$
CONV3-10		$28 \times 28 \times 10$
POOL2	0	$14 \times 14 \times 10$
CONV3-10	$3 \times 3 \times 10 \times 10 + 10$	
POOL2		
FLATTEN	0	490
FC-3		3

(f) Consider a new architecture:

$$\text{CONV2-3} \rightarrow \text{ReLU} \rightarrow \text{CONV2-3} \rightarrow \text{ReLU} \rightarrow \text{GAP (Global Average Pool)} \rightarrow \text{FC-3}$$

Each CONV2-3 layer has stride of 1 and padding of 1. Note that we use **circular padding** (i.e. wrap-around) for this task. Instead of using zeros, circular padding makes it as though the virtual column before the first column is the last column and the virtual row before the first row is the last row — treating the image as though it was on a torus.

Here, the GAP layer is an average pooling layer that computes the per-channel means over the entire input image.

You are told the behavior for an input image with a horizontal edge, $\mathbf{x}_1$ and an image with a vertical edge, $\mathbf{x}_2$:

$$\mathbf{x}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{x}_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Suppose we knew that the GAP output features when fed $\mathbf{x}_1$ and $\mathbf{x}_2$ are

$$\mathbf{g}_1 = f(\mathbf{x}_1) = \begin{bmatrix} 0.8 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{g}_2 = f(\mathbf{x}_2) = \begin{bmatrix} 0 \\ 0.8 \\ 0 \end{bmatrix}$$

Use what you know about the invariances/equivariances of convolutional nets to compute the $\mathbf{g}_i$ corresponding to the following $\mathbf{x}_i$ images.

- $\mathbf{x}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

- $\mathbf{x}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

3. Convolutional Networks

Note: Throughout this problem, we will use the convention of NOT flipping the filter before dragging it over the signal. This is the standard notation with neural networks (ie, we assume the filter given to us is already flipped)

(a) **List two reasons we typically prefer convolutional layers instead of fully connected layers when working with image data.**

(b) Consider the following 1D signal: $[1, 4, 0, -2, 3]$. After convolution with a length-3 filter, no padding, stride=1, we get the following sequence: $[-2, 2, 11]$. **What was the filter?**
(Hint: Just to help you check your work, the first entry in the filter that you should find is 2. However, if you try to use this hint directly to solve for the answer, you will not get credit since this hint only exists to help you check your work.)

(c) Transpose convolution is an operation to help us upsample a signal (increase the resolution). For example, if our original signal were $[a, b, c]$ and we perform transpose convolution with pad=0 and stride=2, with the filter $[x, y, z]$, the output would be $[ax, ay, az + bx, by, bz + cx, cy, cz]$. Notice that the entries of the input are multipled by each of the entries of the filter. Overlaps are summed. Also notice how for a fixed filtersize and stride, the dimensions of the input and output are swapped compared to standard convolution. (For example, if we did standard convolution on a length-7 sequence with filtersize of 3 and stride=2, we would output a length-3 sequence).
If our 2D input is $\begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}$ and the 2D filter is $\begin{bmatrix} +1 & -1 \\ 0 & +1 \end{bmatrix}$ **What is the output of transpose convolution with pad=0 and stride=1?**

4. Convolutional Networks and Dilated Convolutions

(a) Consider a convolutional neural network layer with:

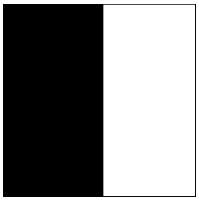
- Input shape: [10, 3, 32, 32] (batch size, number of input channels, input height, input width)
- Number of filters: 64

For each of the following configurations of kernel size, stride, and padding, **calculate the output dimensions** $[n, c, h, w]$ where n is the batch size, c is the number of channels, and h, w are the output height and width. Assume that each layer has the same input shape and number of filters as specified above.

i. Kernel size: 3x3, stride: 1, padding: 1

ii. Kernel size: 4x4, stride: 2, padding: 0

(b) **Design a 3x3 filter that detects vertical edges like the one shown in the image below.**



$$\begin{bmatrix} \_ & \_ & \_ \\ \_ & \_ & \_ \\ \_ & \_ & \_ \end{bmatrix}$$

(c) **Design a 3x3 filter to blur an image.**
(Hint: blurring involves averaging a pixel's value with those of its neighbors.)

$$\begin{bmatrix} \_ & \_ & \_ \\ \_ & \_ & \_ \\ \_ & \_ & \_ \end{bmatrix}$$

(d) In a regular convolutional operation, the kernel slides over the input data in a contiguous manner. However, in dilated convolution, the kernel is "dilated" by introducing gaps between its elements, resulting in a larger receptive field for each output pixel.

The dilation (d) determines the spacing between kernel elements — a dilation of d introduces $d - 1$ gaps between two kernel elements. The example above illustrates a 3x3 1-dilated kernel ($d = 1$ is equivalent to a regular convolutional kernel) and a 3x3 2-dilated ($d = 2$) kernel.

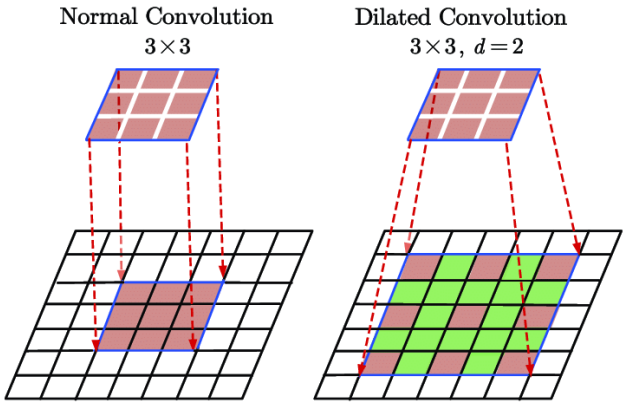


Figure 1: Dilated convolution kernels. Source: "Brain MRI Super-Resolution Using 3D Dilated Convolutional Encoder–Decoder Network" by Du et al.

i. You are given an input matrix M and 2x2 filter k below. **Compute their dilated convolution with $d = 2$.** Assume that gaps in the kernel are filled with zeros, stride is 1 and padding is 0.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad k = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

ii. Consider a two-layer architecture:

`DilatedConv1(3x3, d=1)` $\rightarrow$ `DilatedConv2(3x3, d=2)`

Both layers use the same sized kernel (3x3), but `DilatedConv1` has a dilation $d = 1$ and `DilatedConv2` has a dilation $d = 2$. **Compute the size of the receptive field at the output of the final layer (DilatedConv2).**

Recall that the receptive field is the region in the *original input* image whose pixels affect the output for a pixel in the specified layer.

5. Weights and Gradients in a CNN

In this homework assignment, we aim to accomplish two objectives. Firstly, we seek to comprehend that the weights of a CNN are a weighted average of the images in the dataset. This understanding is crucial in answering a commonly asked question: does a CNN memorize images during the training process? Additionally, we will analyze the impact of spatial weight sharing in convolution layers. Secondly, we aim to gain an understanding of the behavior of max-pooling and avg-pooling in backpropagation. By accomplishing these objectives, we will enhance our knowledge of CNNs and their functioning.

Let's consider a convolution layer with input matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$,

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,n} \end{bmatrix}, \tag{1}$$

weight matrix $\mathbf{w} \in \mathbb{R}^{k \times k}$,

$$\mathbf{w} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \cdots & w_{k,k} \end{bmatrix}, \tag{2}$$

and output matrix $\mathbf{Y} \in \mathbb{R}^{m \times m}$,

$$\mathbf{Y} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,m} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m,1} & y_{m,2} & \cdots & y_{m,m} \end{bmatrix}. \tag{3}$$

For simplicity, we assume the number of the input channel (of $\mathbf{X}$ is) and the number of the output channel (of output $\mathbf{Y}$) are both 1, and the convolutional layer has no padding and a stride of 1.

Then for all i, j,

$$y_{i,j} = \sum_{h=1}^{k} \sum_{l=1}^{k} x_{i+h-1,j+l-1} w_{h,l}, \tag{4}$$

or

$$\mathbf{Y} = \mathbf{X} * \mathbf{w}, \tag{5}$$

, where $*$ refers to the convolution operation. For simplicity, we omitted the bias term in this question.

Suppose the final loss is $\mathcal{L}$, and the upstream gradient is $d\mathbf{Y} \in \mathbb{R}^{m,m}$,

$$d\mathbf{Y} = \begin{bmatrix} dy_{1,1} & dy_{1,2} & \cdots & dy_{1,m} \\ dy_{2,1} & dy_{2,2} & \cdots & dy_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ dy_{m,1} & dy_{m,2} & \cdots & dy_{m,m} \end{bmatrix}, \tag{6}$$

where $dy_{i,j}$ denotes $\dfrac{\partial \mathcal{L}}{\partial y_{i,j}}$.

(a) **Derive the gradient to the weight matrix** $d\mathbf{w} \in \mathbb{R}^{k,k}$,

$$d\mathbf{w} = \begin{bmatrix} dw_{1,1} & dw_{1,2} & \cdots & dw_{1,k} \\ dw_{2,1} & dw_{2,2} & \cdots & dw_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ dw_{k,1} & dw_{k,2} & \cdots & dw_{k,k} \end{bmatrix}, \tag{7}$$

where $dw_{h,l}$ denotes $\dfrac{\partial \mathcal{L}}{\partial w_{h,l}}$. Also, **derive the weight after one SGD step with a batch of a single image**.

(b) The objective of this part is to investigate the effect of spatial weight sharing in convolution layers on the behavior of gradient norms with respect to changes in image size.
For simplicity of analysis, we assume $x_{i,j}, dy_{i,j}$ are independent random variables, where for all i, j:

$$\mathbb{E}[x_{i,j}] = 0, \tag{8}$$
$$\mathrm{Var}(x_{i,j}) = \sigma_x^2, \tag{9}$$
$$\mathbb{E}[dy_{i,j}] = 0, \tag{10}$$
$$\mathrm{Var}(dy_{i,j}) = \sigma_g^2. \tag{11}$$

Derive the mean and variance of $dw_{h,l} = \dfrac{\partial \mathcal{L}}{\partial W_{h,l}}$ **for each** i, j **a function of** n, k, σ_x, σ_g**. What is the asymptotic growth rate of the standard deviation of the gradient on** $dw_{h,l}$ **with respect to the length and width of the image** n**?**
Hint: there should be no m *in your solution because* m *can be derived from* n *and* k*.*
Hint: you cannot assume that $x_{i,j}$ *and* $dy_{i,j}$ *follow normal distributions in your derivation or proof.*

(c) **For a network with only 2x2 max-pooling layers (no convolution layers, no activations), what will be** $d\mathbf{X} = [dx_{i,j}] = [\dfrac{\partial \mathcal{L}}{\partial x_{i,j}}]$**? For a network with only 2x2 average-pooling layers (no convolution layers, no activations), what will be** $d\mathbf{X}$**?**
HINT: Start with the simplest case first, where $\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$*. Further assume that top left value is selected by the max operation. i.e.*

$$y_{1,1} = x_{1,1} = \max(x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2}) \tag{12}$$

Then generalize to higher dimension and arbitrary max positions.

(d) Following the previous part, **discuss the advantages of max pooling and average pooling** in your own words.
Hint: you may find it helpful to finish the question "Inductive Bias of CNNs (Coding Question)" before working on this question

6. Inductive Bias of CNNs (Coding Question)

In this problem, you will follow the EdgeDetection.ipynb notebook to understand the inductive bias of CNNs.

(a) Overfitting Models to Small Dataset: **Fill out notebook section (Q1).**

(i) Can you find any interesting patterns in the learned filters?

(b) Sweeping the Number of Training Images: **Fill out notebook section (Q2).**

(i) Compare the learned kernels, untrained kernels, and edge-detector kernels. What do you observe?

(ii) We freeze the convolutional layer and train only final layer (classifier). In a high data regime, the performance of CNN initialized with edge detectors is worse than CNN initialized with random weights. Why do you think this happens?

(c) Checking the Training Procedures: **Fill out notebook section (Q3).**

(i) List every epochs that you trained the model. Final accuracy of CNN should be at least 90% for 20 images per class.

(ii) Check the learned kernels. What do you observe?

(iii) (optional) You might find that with the high number of epochs, validation loss of MLP is increasing whild validation accuracy increasing. How can we interpret this?

(iv) (optional) Do hyperparameter tuning. And list the best hyperparameter setting that you found and report the final accuracy of CNN and MLP.

(v) How much more data is needed for MLP to get a competitive performance with CNN? Does MLP really generalize or memorize?

(d) Domain Shift between Training and Validation Set: **Fill out notebook section (Q4).**

(i) Why do you think the confusion matrix looks like this? Why CNN misclassifies the images with edge to the images with no edge? Why MLP misclassifies the images with vertical edge to the images with horizontal edge and vice versa? (Hint: Visualize some of the images in the training and validation set.)

(ii) Why do you think MLP fails to learn the task while CNN can learn the task? (Hint: Think about the model architecture.)

(e) When CNN is Worse than MLP: **Fill out notebook section (Q5).**

(i) What do you observe? What is the reason that CNN is worse than MLP? (Hint: Think about the model architecture.)

(ii) Assuming we are increasing kernel size of CNN. Does the validation accuracy increase or decrease? Why?

(iii) How do the learned kernels look like? Explain why.

(f) Increasing the Number of Classes: **Fill out notebook section (Q6).**

(i) Compare the performance of CNN with max pooling and average pooling. What are the advantages of each pooling method?

7. Memory considerations when using GPUs (Coding Question)

In this homework, you will run GPUMemory.ipynb to train a ResNet model on CIFAR-10 using PyTorch and explore its implications on GPU memory.

We will explore various systems considerations, such as the effect of batch size on memory usage and how different optimizers (SGD, SGD with momentum, Adam) vary in their memory requirements.

It is strongly recommended that you start early on this question, since colab daily GPU limits may require you to complete this question over a few days with breaks in between.

(a) Managing GPU memory for training neural networks (Notebook Section 1).
   (i) **How many trainable parameters does ResNet-152 have**? What is the **estimated size of the model in MB**?
   (ii) Which GPU are you using? **How much total memory does it have**?
   (iii) After you load the model into memory, **what is the memory overhead (MB) of the CUDA context loaded with the model**?

(b) Optimizer memory usage (Notebook Section 2).
   (i) **What is the total memory utilization during training with SGD, SGD with momentum and Adam optimizers**? Report in MB individually for each optimizer.
   (ii) **Which optimizer consumes the most memory**? Why?

(c) Batch size, learning rates and memory utilization (Notebook Section 3)
   (i) **What is the memory utilization for different batch sizes (4, 16, 64, 256)? What is the largest batch size you were able to train?**
   (ii) **Which batch size gave you the highest accuracy at the end of 10 epochs?**
   (iii) **Which batch size completed 10 epochs the fastest (least wall clock time)? Why?**
   (iv) Attach your **training accuracy vs wall time plots** with your written submission.

8. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**
(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)
(c) **Roughly how many total hours did you work on this homework?**

Contributors:

- Sheng Shen.
- Suhong Moon.
- Jake Austin.
- Kevin Li.
- Anant Sahai.
- Fei-Fei Li.
- Saagar Sanghavi.
- Romil Bhardwaj.
- Peter Wang.
- Linyuan Gong.
- Long He.
- Kumar Krishna Agrawal.

