CS4803-7643: Deep Learning
Spring 2022
Problem Set 3

Instructor: Zsolt Kira
TAs: Bhavika Devnani, Jordan Rodrigues, Mandy Xie,
Yanzhe Zhang, Amogh Dabholkar, Ahmed Shaikh,
Ting-Yu Lan, Anshul Ahluwalia, Aditya Singh, Yash Jakhotiya

Discussions: `https://piazza.com/gatech/spring2022/cs46447643a`

Due: Monday, March 14, 11:59pm EST

Instructions

1. We will be using Gradescope to collect your assignments. Please read the following instructions for submitting to Gradescope carefully!

    - For the HW3 component on Gradescope, you could upload one single PDF containing the answers to all the theory questions, the report for the coding problems and the jupyter notebook *root/test_style_transfer.ipynb* consisting of tests for parts of the style transfer implementation . However, the solution to each problem/subproblem must be on a separate page. When submitting to Gradescope, please make sure to mark the page(s) corresponding to each problem/sub-problem. Also, please make sure that your submission for the coding part only includes the files collected by the collect_submission script, else the auto-grader will result in a zero, and we won't accept regrade requests for this scenario given the size of the class. Likewise, the pages of the report must also be marked to their corresponding subproblems
    - For the coding problem, please use the provided `collect_submission.py` script and upload the resulting zip file to the HW3 Code assignment on Gradescope. Only Style Transfer will be graded by the autograder. Please make sure you have saved the most recent version of your code before running this script.
    - Note: This is a large class and Gradescope's assignment segmentation features are essential. Failure to follow these instructions may result in parts of your assignment not being graded. We will not entertain regrading requests for failure to follow instructions. Please check this link for additional information on submitting to Gradescope.

2. LaTeX'd solutions are strongly encouraged (solutions template is provided in the starter zip file) but scanned handwritten copies are acceptable. Hard copies are **not** accepted.

3. We generally encourage you to collaborate with other students.

    You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and *not* as a group activity. Please list the students you collaborated with.

1 Collaborators [0.5 points]

Please list your collaborators and assign this list to the corresponding section of the outline on Gradescope. If you don't have any collaborators, please write 'None' and assign it to the corresponding section of the Gradescope submission regardless.

2 Convolution Basics

1. **[0.5 points]** The convolution layer inside of a CNN is intrinsically an *affine transformation*: A vector is received as input and is multiplied with a matrix to produce an output (to which a bias vector is usually added before passing the result through a nonlinearity). This operation can be represented as $y = Ax$, in which A describes the affine transformation.

   We will first revisit the convolution layer as discussed in the class. Consider a convolution layer with a 3x3 kernel W, operated on a single input channel X, represented as:

$$W = \begin{bmatrix} w_{(0,0)}, w_{(0,1)}, w_{(0,2)} \\ w_{(1,0)}, w_{(1,1)}, w_{(1,2)} \\ w_{(2,0)}, w_{(2,1)}, w_{(2,2)} \end{bmatrix}, X = \begin{bmatrix} x_{(0,0)}, x_{(0,1)}, x_{(0,2)} \\ x_{(1,0)}, x_{(1,1)}, x_{(1,2)} \\ x_{(2,0)}, x_{(2,1)}, x_{(2,2)} \end{bmatrix} \tag{1}$$

   Now let us work out a **stride-4** convolution layer, with **zero padding size of 2**. Consider 'flattening' the input tensor X in row-major order as:

$$X = \left[x_{(0,0)}, x_{(0,1)}, x_{(0,2)}, ..., x_{(2,0)}, x_{(2,1)}, x_{(2,2)}\right]^\top \tag{2}$$

   Write down the convolution as a matrix operation A such that: $Y = AX$. Output Y is also flattened in row-major order.

2. **[0.5 points]** Recall that transposed convolution can help us upsample the feature size spatially. Consider a transposed convolution layer with a 2x2 kernel W operated on a single input channel X, represented as:

$$W = \begin{bmatrix} w_{(0,0)}, w_{(0,1)} \\ w_{(1,0)}, w_{(1,1)} \end{bmatrix}, X = \begin{bmatrix} x_{(0,0)}, x_{(0,1)} \\ x_{(1,0)}, x_{(1,1)} \end{bmatrix} \tag{3}$$

   We 'flattened' the input tensor in row-major order as $X = [x_{(0,0)}, x_{(0,1)}, x_{(1,0)}, x_{(1,1)}]$.

   Write down the affine transformation A corresponding to a transposed convolution layer with kernel W, **stride 2, no padding**. Output Y is also flattened in row-major order.

3. **[1 point]** Convolution layers in most CNNs consist of multiple input and output feature maps. The collection of kernels form a 4D tensor (output channels o, input channels i, filter rows k, filter columns k), represented in short as (o, i, k, k). For each output channel, each input

channel is convolved with a distinct 2D slice of the 4D kernel and the resulting set of feature maps is summed element-wise to produce the corresponding output feature map.

There is an interesting property that a convolutional layer with kernel size $(o \times r^2, i, k, k)$ is identical to a transposed convolution layer with kernel size $(o, i, k \times r, k \times r)$. Here the word 'identical' means with the same input feature X, both operations will give the same output Y with only a difference in the ordering of flattened elements of Y.

Now let us prove the property in a restricted setting. Consider $o = 1, r = 2, i = 1, k = 1$. Given the same input feature X as in equation 3, write down the affine transformation for a convolutional layer with kernel size (4, 1, 1, 1), and show that it is an operation identical to a transposed convolution layer with kernel size $(1, 1, 2, 2)$. Use whatever parameters for stride and padding that make the output Y the same size (size meaning number of elements, not the same shape). Y should have 16 elements.

3 Logic and XOR

4. **[1 point]** Implement AND and OR for pairs of binary inputs using a single linear threshold neuron with weights $\mathbf{w} \in \mathbb{R}^2$, bias $b \in \mathbb{R}$, and $\mathbf{x} \in \{0, 1\}^2$:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ 0 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases} \tag{4}$$

That is, find $\mathbf{w}_{\texttt{AND}}$ and $b_{\texttt{AND}}$ such that

x_1	x_2	$f_{\texttt{AND}}(\mathbf{x})$
0	0	0
0	1	0
1	0	0
1	1	1

Also find $\mathbf{w}_{\texttt{OR}}$ and $b_{\texttt{OR}}$ such that

x_1	x_2	$f_{\texttt{OR}}(\mathbf{x})$
0	0	0
0	1	1
1	0	1
1	1	1

5. **[1 point]** Consider the XOR function

x_1	x_2	$f_{\texttt{XOR}}(x)$
0	0	0
0	1	1
1	0	1
1	1	0

Prove that XOR can NOT be represented using a linear model with the same form as (4).

[*Hint:* To see why, plot the examples from above in a plane and think about drawing a linear boundary that separates them.]

4 Piece-wise Linearity

6. **[3 points]** Consider a specific 2 hidden layer ReLU network with inputs $x \in \mathbb{R}$, 1 dimensional outputs, and 2 neurons per hidden layer. This function is given by

$$h(x) = W^{(3)} \max\{0, W^{(2)} \max\{0, W^{(1)}x + \mathbf{b}^{(1)}\} + b^{(2)}\} + b^{(3)} \tag{5}$$

with weights:

$$W^{(1)} = \begin{bmatrix} 1.5 \\ 0.5 \end{bmatrix} \tag{6}$$

$$b^{(1)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{7}$$

$$W^{(2)} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \tag{8}$$

$$b^{(2)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{9}$$

$$W^{(3)} = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{10}$$

$$b^{(3)} = -1 \tag{11}$$

An interesting property of networks with piece-wise linear activations like the ReLU is that on the whole they compute piece-wise linear functions. At each of the following points $x = x_o$, determine the value of weight $W \in \mathbb{R}$ and bias $b \in \mathbb{R}$ such that $\frac{dh(x)}{dx}|_{x=x_o} = W$ and $Wx_o + b = h(x_o)$.

$$x_o = 2 \tag{12}$$

$$x_o = -1 \tag{13}$$

$$x_o = 1 \tag{14}$$

5 Depth - Composing Linear Pieces

7. **[3 points]** Now we'll turn to a more recent result that highlights the *Deep* in Deep Learning. Depth (composing more functions) results in a favorable combinatorial explosion in the "number of things that a neural net can represent". For example, to classify a cat it seems useful to first find parts of a cat: eyes, ears, tail, fur, *etc*. The function which computes a probability of cat presence should be a function of these components because this allows everything you learn about eyes to generalize to all instances of eyes instead of just a single instance. Below you will detail one formalizable sense of this combinatorial explosion for a particular class of piecewise linear networks.

Consider $y = \sigma(x) = |x|$ for scalar $x \in \mathcal{X} \subseteq \mathbb{R}$ and $y \in \mathcal{Y} \subseteq \mathbb{R}$ (Fig. 1). It has one linear region on $x < 0$ and another on $x > 0$ and the activation identifies these regions, mapping both of them to $y > 0$. More precisely, *for each linear region of the input*, $\sigma(\cdot)$ is a bijection. There is

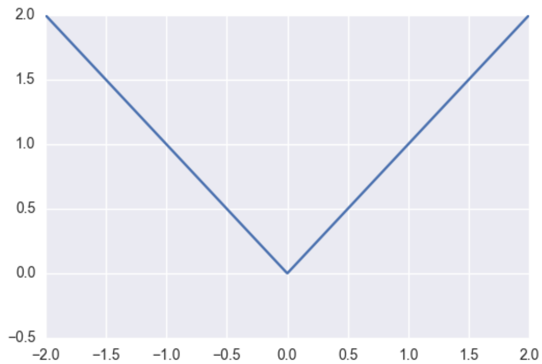

Figure 1

a mapping to and from the output space and the corresponding input space. However, given an output y, it's impossible to tell which linear region of the input it came from, thus $\sigma(\cdot)$ *identifies* (maps on top of each other) the two linear regions of its input. This is the crucial definition because when a function identifies multiple regions of its domain that means any subsequent computation applies to all of those regions. When these regions come from an input space like the space of images, functions which identify many regions where different images might fall (*e.g.*, slightly different images of a cat) automatically transfer what they learn about a particular cat to cats in the other regions.

More formally, we will say that $\sigma(\cdot)$ identifies a set of M disjoint input regions $\mathcal{R} = \{R_1, \ldots, R_M\}$ (*e.g.*, $\mathcal{R} = \{(-1, 0), (0, 1)\}$) with $R_i \subseteq \mathcal{X}$ onto one output region $O \subseteq \mathcal{Y}$ (*e.g.*, $(0, 1)$) if for all $R_i \in \mathcal{R}$ there is a bijection from R_i to O. [1]

Start by applying the above notion of identified regions to linear regions of one layer of a particular neural net that uses absolute value functions as activations. Let $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{y} \in \mathbb{R}^d$ [2], and pick weights $W^{(1)} \in \mathbb{R}^{d \times d}$ and bias $\mathbf{b}^{(1)} \in \mathbb{R}^d$ as follows:

$$W_{ij}^{(1)} = \begin{cases} 2 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} b_i^{(1)} = -1 \tag{15}$$

Then one layer of a neural net with absolute value activation functions is given by

$$f_1(\mathbf{x}) = |W^{(1)}\mathbf{x} + \mathbf{b}| \tag{16}$$

Note that this is an absolute value function applied piece-wise and not a norm.

How many regions of the input are identified onto $O = (0, 1)^d$ by $f_1(\cdot)$? Prove it.

[3]

[1] Recall that a bijection from X to Y is a function $\mu : X \to Y$ such that for all $y \in Y$ there exists a **unique** $x \in X$ with $\mu(x) = y$.

[2] Outputs are in some feature space, not a label space. Normally a linear classifier would be placed on top of what we are here calling $\mathbf{y}$.

[3] Absolute value activations are chosen to make the problem simpler, but a similar result holds for ReLU units. Also, O could be the positive orthant (unbounded above).

8. **[2 points]** Next consider what happens when two of these functions are composed. Suppose g identifies n_g regions of $(0,1)^d$ onto $(0,1)^d$ and f identifies n_f regions of $(0,1)^d$ onto $(0,1)^d$. How many regions of its input does $f \circ g(\cdot)$ identify onto $(0,1)^d$?

9. **[4 points]** Finally consider a series of L layers identical to the one in question 4.1.

$$\mathbf{h}_1 = |W_1 \mathbf{x} + \mathbf{b}_1| \tag{17}$$
$$\mathbf{h}_2 = |W_2 \mathbf{h}_1 + \mathbf{b}_2| \tag{18}$$
$$\vdots \tag{19}$$
$$\mathbf{h}_L = |W_L \mathbf{h}_{L-1} + \mathbf{b}_L| \tag{20}$$

   Let $\mathbf{x} \in (0,1)^d$ and $f(\mathbf{x}) = \mathbf{h}_L$. Note that each $\mathbf{h}_i$ is *implicitly* a function of $\mathbf{x}$. Show that $f(\mathbf{x})$ identifies 2^{Ld} regions of its input.

Conclusion to the above questions

Now compare the number of identified regions for an L layer net to that of an $L-1$ layer net. The L layer net can separate its input space into 2^d more linear regions than the $L-1$ layer net. On the other hand, the number of parameters and the amount of computation time grows linearly in the number of layers. In this very particular sense (which doesn't always align well with practice) deeper is better.

To summarize this problem set, you've shown a number of results about the representation power of different neural net architectures. First, neural nets (even single neurons) can represent logical operations. Second, neural nets we use today compute piece-wise linear functions of their input. Third, the representation power of neural nets increases exponentially with the number of layers.

The point of the exercise was to convey intuition that removes some of the magic from neural nets representations. Specifically, neural nets can decompose problems logically, and piece-wise linear functions can be surprisingly powerful.

6 Implicit Regularization of Gradient descent [Extra Credit for both 4803 and 7643, 6 points]

10. **[4 points]** Classical results in statistical learning theory suggest that as the model class becomes larger, we should expect more overfitting, with everything else staying fixed (such as the training dataset size). However, this classical view appears to be at odds with the 'street wisdom' among practitioners (particularly in deep learning) that *'larger models are better'*. A gem sometimes expressed as *'add MOAR layers!'*. More formally, in deep learning it is fairly common to have more parameters in the model than number of datapoints ($d > n$). Results from classical statistics will tell you that such settings are hopeless without more assumptions. In this section and the next one, we will try to reconcile this ostensible disagreement.

    Specifically, in this question we will build intuition using a simple least-squares linear regression problem, optimized with gradient descent.

Consider a linear regression problem with a d-dimensional feature vector $\mathbf{x} \in \mathbb{R}^d$ as input and $y_i \in \mathbb{R}$ as the output. The dataset consists of n training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), \ldots, (\mathbf{x}_n, y_n)$. Let $\mathbf{X}$ be the $n \times d$ data matrix formed by placing the feature vectors on the rows of this matrix,

i.e. $\mathbf{X}^T = \left[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\right]$.

Let $\mathbf{y}$ be a column vector with elements $y_1, y_2 \ldots y_n$. We will operate in the setting where $d > n$, *i.e.* there are more feature dimensions than samples. Thus, the following linear system is under-constrained / under-determined:

$$\mathbf{X}\mathbf{w} = \mathbf{y} \tag{21}$$

To avoid the degenerate case, we will assume that $\mathbf{y}$ lies in the span of $\mathbf{X}$, *i.e.* this linear system has at least one solution.

Now, recall that the optimization problem in least-squares regression is the following:

$$\min_{\mathbf{w} \in \mathbf{R}^\mathbf{d}} \quad f(\mathbf{w}) = \sum_{i=1}^{n} \underbrace{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}_{\text{squared error on example } i} \tag{22}$$

We will optimize (22) via gradient descent. Specifically, let us initialize $\mathbf{w}^{(0)} = 0$. And repeatedly take a step in the direction of negative gradient with a sufficiently-small constant step-size η till convergence:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)}) \tag{23}$$

Let us refer to the solution found by gradient descent at convergence as $\mathbf{w}^{gd}$.

Prove that the solution found by gradient descent for least-squares is equal to the result of the following *different* optimization problem:

$$\mathbf{w}^{gd} = \underset{\mathbf{w} \in \mathbf{R}^\mathbf{d}}{\text{argmin}} \quad ||\mathbf{w}||_2^2 \tag{24}$$

$$s.t. \quad \mathbf{X}\mathbf{w} = \mathbf{y} \tag{25}$$

11. **[1 point]** In a few words, describe what the optimization problem in (24), (25) is trying to express. Specifically, explain what the constraint (25) means/represents in terms of training error of least-squares. Explain what the objective (24) means in the context of machine learning. Putting them together, what solution is (24), (25) looking for.

12. **[1 point]** What does this tell you about gradient descent (in the context of least squares regression)? Notice that (22) is an unconstrained optimization problem. Is the solution unique? Does gradient descent appear to have a preference among the solutions?

7 Receptive Fields [Extra Credit for both 4803 and 7643, 3 points]

13. In Convolutional Neural Networks (CNNs), the Receptive Field (RF) of a pixel in a feature map is defined as the size of the region in the CNN input that plays a role in the computation of the feature-map pixel. RF is used to measure the association of an output feature to the input region.

    (a) **[0.5 point]** Can you describe one scenario where we have to be concerned about the receptive field?

    Now that we understand the importance of Receptive Fields, let's calculate the Receptive Field for a Fully-Convolutional Network (FCN). For this question, we will be using 1-dimensional input signals and feature maps, but the derivations can be applied to each dimension independently for higher-dimensional signals. Consider a FCN with L layers, $l = 1, 2, \ldots, L$, where each layer l is parameterized by 3 variables, k_l: kernel size, s_l: stride, and p_l: padding. The feature map $f_l \in R^{h_l \times w_l}$ is the output of the l-th layer, with height h_l and width w_l, and the input image is denoted by f_0. As discussed in class, we can calculate h_l and w_l of feature map f_l using equation (26) and (27) respectively

$$h_l = \left(\frac{h_{l-1} + 2p_l - k_l}{s_l}\right) + 1 \tag{26}$$

$$w_l = \left(\frac{w_{l-1} + 2p_l - k_l}{s_l}\right) + 1 \tag{27}$$

    Let r_l correspond to the number of features in the feature map f_l which contributes to generate one feature in f_L. Let's take a scenario where we know r_l and want to compute r_{l-1}. Each feature from f_l is connected to k_l features from f_{l-1}. We will keep things simple by setting $k_l = 1$; in this case, as illustrated in figure 2, r_l features in f_l will cover $r_{l-1} = r_l \cdot s_l - (s_l - 1)$, $r_l \cdot s_l$ covers the entire region where the features come from, but has overlap with $(s_l - 1)$ features.

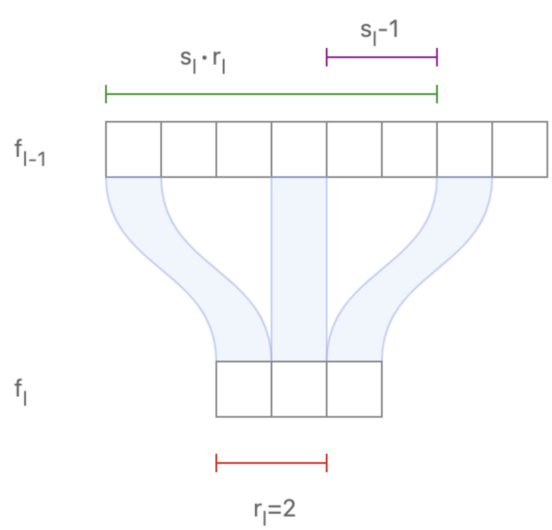

Figure 2

    Let's take a more general scenario where $k_l > 1$, in addition to $r_l \cdot s_l - (s_l - 1)$, we will also have to account for the regions on left and right as illustrated in figure 3. To account for the regions on left and right, we can add $k_l - 1$ to $r_l \cdot s_l - (s_l - 1)$ to obtain the following recurrence relation

$$r_{l-1} = r_l \cdot s_l + (k_l - s_l) \tag{28}$$

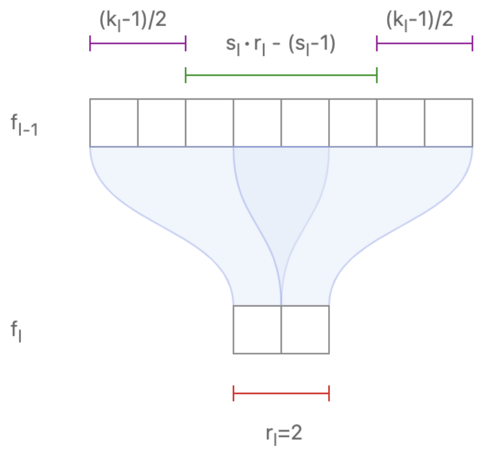

Figure 3

The above recurrence relation can be solved to obtain a solution in terms of kernel size and stride as shown in equation(29), which will let us compute the Receptive Field size of the network r_0.

$$r_0 = \sum_{l=1}^{L} \left((k_l - 1) * \prod_{i=1}^{l-1} s_i \right) + 1 \tag{29}$$

Let's take a FCN with one convolutional layer with parameters $k_1 = 11$, $s_1 = 4$, and $p_1 = 0$, ReLU and Maxpooling layer with parameters $k_2 = 3$, $s_2 = 2$, and $p_2 = 0$, and input $f_0 = 259$.

(b) **[1 point]** Calculate Receptive Fields of a pixel in the feature maps f_1 and f_2.

From the calculations, we can notice that each feature in the feature map f_2 is produced by a small input region and may not contain all the necessary information. Let's try to increase the Receptive Fields, so the FCN will incorporate more details from the input. One of the techniques to increase the Receptive Field is to add more convolutional layers in the network. Let's add more layers with the following spatial configuration
configuration

Conv2 Layer:	$k_3 = 5,\ s_3 = 1,\ p_3 = 2$
maxpool Layer:	$k_4 = 3,\ s_4 = 2,\ p_4 = 0$
Conv3 Layer:	$k_5 = 3,\ s_5 = 1,\ p_5 = 1$
Conv4 Layer:	$k_6 = 3,\ s_6 = 1,\ p_6 = 1$
Conv5 Layer:	$k_7 = 3,\ s_7 = 1,\ p_7 = 1$

(c) **[1.5 points]** Calculate the size of the feature maps and Receptive Fields for a pixel in each feature map for the added layers.

Now that we have established adding layers can increase the receptive fields, we can start thinking about the contribution of each pixels in a receptive field to an output unit's response. The relative importance of each input pixel can be defined as the Effective Receptive Field(ERF) of the feature[Ref]. You cannot help but to notice that the pixels at the center of a receptive field has a larger impact on output due to more paths to contribute to the output. I hope this discussion motivates you to explore ways to formalize this intuition and explore more about the Effective Receptive Field(ERF).

8 Paper Review [Extra credit for 4803, regular credit for 7643]

The paper we will study in this homework is **'Deep Double Descent: Where Bigger Models And More Data Hurt'**.

The paper presents a concept called 'double descent' (in the context of deep learning), with interesting experiments showing model performance first getting worse and then improving as we increase model size.

The paper can be viewed here. The evaluation rubric for this section is as follows:

14. **[2 points]** Briefly summarize the key contributions, strengths and weaknesses of this paper.

15. **[2 points]** What is your personal takeaway from this paper? This could be expressed either in terms of relating the approaches adopted in this paper to your traditional understanding of learning parameterized models, or potential future directions of research in the area which the authors haven't addressed, or anything else that struck you as being noteworthy.

Guidelines: Please restrict your reviews to no more than 350 words (total length for answers to both the above questions).

9 Coding: Uses of Gradients With Respect to Input

16. **[24 points for both sections]** The coding part of this assignment will have you implement different ideas which all rely on gradients of some neural net output with respect to the input image. To get started, download the zip file (the one containing code) from Canvas. Also, be sure to append the completed write up (available in the zip folder) and the jupyter notebook *root/test_style_transfer.ipynb* (as mentioned in the instructions) to the PS3 solutions and upload it on Gradescope under HW3 writeup.