CIS 620, Advanced Topics in Deep Learning, Spring 2024

Homework 2
Due: Monday, February 5, 11:59pm
Submit to Gradescope

Learning Objectives

After completing this assignment, you will:

- Build a suitable vision model
- Experiment with hyperparameters such as filter size
- Learn about visualizing layers of vision models

Deliverables

This is a **pair** assignment for both the written and coding portions. - **One submission per pair**

1. **A PDF with your name in the agreement**

   Copy and edit this googledoc to enter your names in the Student Agreement and answer the questions mentioned below. **Don't forget to add your team member on gradescope submission!**

2. **hw2.ipynb file with the functional code**

   Complete the coding assignment in the Jupyter Notebook and upload the file to Gradescope. **Don't forget to add your team member on the gradescope submission!**

Note that there is a separate assignment for the papers, which will be handed in separately.

Homework Submission Instructions

Written Homeworks

All written homework must be submitted as a PDF to Gradescope. **Handwritten assignments (scanned or otherwise) will not be accepted.**

Coding Homeworks

All coding assignments will be done in Jupyter Notebooks. We will provide a .ipynb template for each assignment as well as function stubs for you to implement. You are free to use your own installation of Jupyter for the assignments, or Google Colab, which provides a Jupyter Environment connected to Google Drive along with a hosted runtime containing a CPU and GPU.

For this assignment, we have provided the code for dataset creation from a single image. The input image is provided in the folder and can also be accessed by the code for dataset creation.

Questions

1. **Design a network that can take as input the scrambled images and produce as output the unscrambled original image.**
    a. **What will be the input to the model? One choice is the normalized scrambled image.**
    b. **What will the model output? Here are a few ways to think of this:**
        i. **An end-to-end architecture that gives as output the unscrambled original image.**
        ii. **A discretization such as the shuffling label for each patch of image_size//4. If you go down this path, think of post-processing steps to display the unscrambled image.**
        iii. **A pixel-level prediction of the target location to unscramble.**
    c. **How are you loading the dataset? What mini-batch sizes are you choosing?**
    d. **What architecture did you finally use? Here are two directions you might want to think of:**
        i. **Pre-trained huggingface vision model and fine-tuning it.**
        ii. **Build a model from scratch in pytorch.**
    e. **Elaborate why you chose the filter size and form (e.g., convolutional or pooling) and size of each layer.**
2. **What is the impact of filter sizes on the training loss? Give a plot or table that shows the effect on prediction accuracy of systematically changing the filter size. What size is best? Why?**
3. **What do you think would work better for this task: ConvNets or ViTs? You don't have to try out ViTs but based on the readings what arguments do you think would hold for this task.**

Optional but fun!

4. **Pick two different layers, and show what the layer might be learning using visualization tools such as GradCam. You can read more about GrandCam here:** https://github.com/jacobgil/pytorch-grad-cam

ANSWERS:-

1.a) The input to the model is a patch of the 4x4 image.
1.b) The output of the model is the predicted label of the input patch.
1.c) The dataset is loaded using a custom class “PatchedImageDataset”, which is a subclass of PyTorch's “Dataset” class.
In the “__getitem__” method of “PatchedImageDataset”, each image and its labels are processed to create patches. The image is split into 16 patches (4x4 grid with each patch having a shape of 56x56x3), and a single label is assigned to each patch. This preprocessing is done every time an item is fetched from the dataset. The mini-batch size used is 32.
1.d) A custom convolutional neural network architecture was used to build a model from scratch using PyTorch.

Convolutional Layers:

The model includes three convolutional layers.

The first convolutional layer (`conv1`) takes an input with 3 channels (RGB image) and outputs 16 feature maps, using a kernel size of 3 and padding of 1.

The second layer (`conv2`) takes these 16 feature maps as input and outputs 32 feature maps, again with a kernel size of 3 and padding of 1.

The third layer (`conv3`) takes the 32 feature maps and outputs 64 feature maps, with the same kernel and padding settings.

Activation Function:

After each convolutional layer, a ReLUactivation function is applied.

Pooling Layers:

Following the ReLU activation after each convolutional layer, a max pooling operation is applied with a window size of 2x2 and a stride of 2.

Fully Connected Layers:

The output from the final pooling layer is flattened and passed through two fully connected (linear) layers.

The first linear layer transforms the feature vector to a size of 128.

The second linear layer further transforms this to a vector of size 16, which is the final output size of the model.

Output:

The model outputs a vector of size 16 for each patch. This represents the probabilities for each patch being in one of 16 possible positions in the 4x4 grid.

1.e) The filter size of 3x3 is pretty much standard and is small enough to capture local features but not too small to be computationally intensive. We use maxpooling to capture the important features and to reduce the computational load and the number of parameters by reducing the spatial dimensions of the feature maps. It also introduces translational invariance. The size of each layer is determined based on how we want to capture the complex features and the final output is 16 because we have to associate the patch of the image to a position in the 4x4 grid.

2. For a filter size of 3, I obtained a loss of zero after ten epochs. As I increase the filter size, the loss is not converging to a minimum and I will have to change the learning rate.

Upon experimentation, I noticed that the loss would increase with increasing filter size because it is very important to capture local features in order to match the image patch to its correct label.

3. If we are to use an architecture where we will have to input a whole image that is scrambled and get an unscrambled image as the output, I would wager the ViT to be the better choice as it would associate every part of the image globally with every other part and this will be very useful when it comes to unscrambling an image.