Deep Learning Systems
(ENGR-E 533)
Homework 2

Instructions

Due date: Oct. 25, 2020, 23:59 PM (Eastern)

- Start early if you're not familiar with the subject, TF or PT programming, and LaTeX.
- Do it yourself. Discussion is fine, but code up on your own
- Late policy
  - If the sum of the late hours (throughout the semester) $<$ seven days (168 hours): no penalty
  - If your total late hours is larger than 168 hours, you'll get only 80% of all the late-submitted homework.
- I ask you to use either PyTorch or Tensorflow running on Python 3.
- Submit a `.ipynb` as a consolidated version of your report and code snippets. But the math should be clear with LaTeX symbols and the explanations should be full by using text cells. In addition, submit an `.html` version of your notebook where you embed your sound clips and images. For example, if you have a graph as a result of your code cell, it should be visible in this `.html` version before we run your code. Ditto for the sound examples.

Problem 1: Speech Denoising Using 1D CNN [5 points]

1. As an audio guy it's sad to admit, but a lot of audio signal processing problems can be solved in the time-frequency domain, or an *image version* of the audio signal. You've learned how to do it in the previous homework by using STFT and its inverse process.
2. What that means is nothing stops you from applying a CNN to the same speech denoising problem. In this question, I'm asking you to implement a 1D CNN that does the speech denoising job in the STFT magnitude domain. 1D CNN here means a variant of CNN which does the convolution operation along only one of the axes. In our case it's the frequency axis.
3. Like you did in homework 1 Q2, install/load `librosa`. Take the magnitude spectrograms of the dirty signal and the clean signal $|\boldsymbol{X}|$ and $|\boldsymbol{S}|$.
4. Both in Tensorflow and PyTorch, you'd better transpose this matrix, so that each row of the matrix is a spectrum. Your 1D CNN will take one of these row vectors as an example, i.e. $|\boldsymbol{X}|_{:,i}^{\top}$. Since this is not an RGB image with three channels, nor you'll use any other information than just the magnitude during training, your input image has only one channel (depth-wise). Coupled with your choice of the minibatch size, the dimensionality of your minibatch would be like this: [(batch size) $\times$ (number of channels) $\times$ (height) $\times$ (width)] = $[B \times 1 \times 1 \times 513]$. Note that depending on the implementation of the 1D CNN layers in TF or PT, it's okay to omit the height information. Carefully read the definition of the function you'll use.

5. You'll also need to define the size of the kernel, which will be $1 \times D$, or simply D depending on the implementation (because we know that there's no convolution along the height axis).

6. If you define K kernels in the first layer, the output feature map's dimension will be $[B \times K \times 1 \times (513 - D + 1)]$. You don't need too many kernels, but feel free to investigate. You don't need too many hidden layers, either.

7. In the end, you know, you have to produce an output matrix of $[B \times 513]$, which are the approximation of the clean magnitude spectra of the batch. It's a dimension hard to match using CNN only, unless you take care of the edges by padding zeros (let's not do zero-padding for this homework). Hence, you may want to flatten the last feature map as a vector, and add a regular linear layer to reduce that dimensionality down to 513.

8. Meanwhile, although this flattening-followed-by-linear-layer approach should work in theory, the dimensionality of your flattened CNN feature map might be too large. To handle this issue, we will used the concept we learned in class, *striding*: usually, a stride larger than 1 can reduce the dimensionality after each CNN layer. You could consider this option in all convolutional layers to reduce the size of the feature maps gradually, so that the input dimensionality of the last fully-connected (FC) layer is manageable. Maxpooling, coupled with the striding technique, would be something to consider.

9. Be very careful about this dimensionality, because you have to define the input and output dimensionality of the FC layer in advance. For example, a stride of 2 pixels will reduce the feature dimension down to roughly 50%, though not exactly if the original dimensionality is an odd number.

10. Don't forget to apply the activation function of your choice, at every layer, especially in the last layer.

11. Try whatever optimization techniques you've learned so far.

12. Check on the quality of the test signals you used in homework 1 Q2. If you can't believe in your subjective listening test, you could go ahead and calculate the SNR thing by using the following equation, but you might only be able to do it on your training signals:

$$\text{SNR} = 10 \log_{10} \frac{\sum_t \big(s(t)\big)^2}{\sum_t \big(s(t) - \hat{s}(t)\big)^2}. \tag{1}$$

    Note that the equation is based on the time-domain signals, the clean speech $s(t)$ and the reconstruction $\hat{s}(t)$. If this SNR value is zero, it means that the recovered signal contains some noise that's as loud as the clean source. Therefore, this number should be larger than 0. Another thing to note is that a too large SNR value calculated from the training signals doesn't always mean a good result on the test signal due to the potential overfitting issue. So, you still need to check the quality of the test signal by listening to it.

Problem 2: Speech Denoising Using 2D CNN [5 points]

1. Now that we know the audio source separation problem can be solved in the image representation, nothing stops us from using 2D CNN for this.

2. To this end, let's define our input "image" properly. You extract an image of 20×513 out of the entire STFT magnitude spectrogram (transposed). That's an input sample. Using this your 2D CNN estimates the cleaned-up spectrum that corresponds to the last (20th) input frame:

$$|\boldsymbol{S}^{\top}_{:,t+19}| \approx \mathcal{F}_{\mathrm{CNN}}\big(|\boldsymbol{X}^{\top}_{:,t:t+19}|\big). \tag{2}$$

What it means is, the network takes all 20 current and previous input frames $|\boldsymbol{S}^{\top}_{:,t:t+19}|$ into account to predict the clean spectrum of the current frame, $t + 19$.

3. Your next image will be another 20 frames shifted by one frame:

$$|\boldsymbol{S}^{\top}_{:,t+20}| \approx \mathcal{F}_{\mathrm{CNN}}\big(|\boldsymbol{X}^{\top}_{:,t+1:t+20}|\big), \tag{3}$$

and so on. Therefore, a pair of adjacent images (unless you shuffle the order) will be with 19 overlapped frames. Since your original STFT spectrogram has 2,459 frames, you can create 2,440 such images as your training dataset.

4. Therefore the input to the 2D CNN will be of [(batch size) $\times 1 \times 20 \times 513$].

5. Your 2D CNN should be of course with a kernel whose size along both the width (frequencies) and the height axes (frames) should be larger than 1. Feel free to investigate different sizes.

6. Otherwise, the basic idea must be similar with the 1D CNN case. You'll still need those techniques as well as the FC layer.

7. Report the denoising results in the same way. One thing to note is that your output will be with only 2,440 spectra, and it's lacking the first 19 frames. You can ignore those first few frames when you calculate the SNR of the training results. A better way is to augment your input $\boldsymbol{X}$ with 19 silent frames (some magnitude spectra with very small random numbers) in the beginning to match the dimension. I recommend the latter approach.

Problem 3: Data Augmentation [5 points]

1. CIFAR10 is a pretty straightforward image classification task, that consists of 10 visual object classes.

2. Download them from here[1] and be ready to use it. Both PyTorch and Tensorflow have options to conveniently load them, but I chose to download them directly and mess around because I found it easier.

3. Set aside 5,000 training examples for validation.

4. **Build your baseline CNN classifier.**

   (a) The images need to be reshaped into $32 \times 32 \times 3$ tensor.

   (b) Each pixel is an integer with 8bit encoding (from 0 to 255). Transform them down to a floating point with a range [0, 1]. 0 means a black pixel and 1 is a white one.

   (c) People like to rescale the pixels to [-1, 1] so that the input to the CNN is well centered around 0, instead of 0.5.

[1] https://www.cs.toronto.edu/ kriz/cifar.html

(d) I know you are eager to try out a fancier net architecture, but let's stick to this simple one:

```
1st 2d conv layer:  there are 10 kernels whose size is 5x5x3; stride=1
Maxpooling:  2x2 with stride=2
1st 2d conv layer:  there are 10 kernels whose size is 5x5x10; stride=1
Maxpooling:  2x2 with stride=2
1st fully-connected layer:  [flattened final feature map] x 20
2st fully-connected layer:  20 x 10 Softmax on the 10 classes
```

Let's stick to ReLU for activation and the He initializer.

(e) Train this net with an Adam optimizer with a default initial learning late (i.e. 0.001). Check on the validation accuracy at the end of every epoch. Report your validation accuracy over the epochs as a graph. This is the performance of your baseline system.

5. **Build another classifier using augmented dataset.** Prepare four different datasets out of the original CIFAR10 training set (except for the 5,000 you set aside for validation):

(a) I know you already changed the scale of the pixels from 0—255 to -1—+1. Let's go back to the intermediate range, 0—1.

(b) **Augmented dataset #1**: Brighten every pixel in every image by 10%, e.g., by multiplying 1.1. Make sure though, that they don't exceed 1. For example, you may want to do something like this: `np.minimum(1.1*X, 1)`.

(c) **Augmented dataset #2**: Darken every pixel in every image by 10%, e.g., by multiplying 0.9.

(d) **Augmented dataset #3**: Flip all images horizontally (not upside down). As if they are mirrored.

(e) **Augmented dataset #4**: The original training set.

(f) Merge the four augmented dataset into one gigantic training set. Since there are 45,000 images in the original training set (after excluding the validation set), after the augmentation you have 45,000×4=180,000 images. Each original image has four different versions: brighter, darker, horizontally flipped, and original versions. Note that the four share the same label: a darker frog is still a frog.

(g) Don't forget to scale back to -1—+1.

(h) You better visualize a few images after the augmentation to make sure what you did is correct.

(i) Train a fresh new network with the same architecture, but using this augmented dataset. Record the validation accuracy over the epochs.

6. Overlay the validation accuracy curve from the baseline with the new curve recorded from the augmented dataset. I ran 200 epochs for both experiments and was able to see convincing results (i.e., the data augmentation improves the validation performance).

7. In theory you have to conduct a test run on the test set, but let's forget about it.

Problem 4: Self-Supervised Learning via Pretext Tasks [6 points]

1. Suppose that you have only 50 labeled examples per class for your CIFAR10 classification problem, totaling 500 training images. Presumably it might be tough to achieve a high performance in this situation.

2. Set aside 500 examples from your training set (I chose the last 500 examples).

3. **The pretext task**:

   (a) On the other hand, we will assume that the rest of the 49,500 training examples are *unlabeled*. We will create a bogus classification problem using them. Let this unlabeled examples (or the examples that you disregard their original labels) be "class 0".

   (b) "class 1": Create a new class, by vertically flipping all the images upside down.

   (c) "class 2": Create another class, by rotating the images 90 degree counter-clock wise.

   (d) Now you have three classes, each of which contains 49,500 labeled examples.

   (e) This is not a classification problem one can be serious about, but the idea here is that a classifier that is trained to solve this problem may need to learn some features that are going to be helpful for the original CIFAR10 classification problem.

   (f) Train a network with the same setup/architecture described in Problem 3. In theory you need to validate every now and then to prevent overfitting, but who cares about this dummy problem? Let's forget about it and just run about a hundred epochs.

   (g) Store your model somewhere safe. Both TF and PT provide a nice way to save the net parameters.

4. **The baseline**:

   (a) Train a classifier from scratch on the 500 CIFAR10 dataset you set aside in the begining. Note that they are for the original 10-class classification problem, and you ARE doing the original CIFAR10 classification, except that you use a ridiculously small amount of dataset. Let's stick to the same architecture/setup. You may need to choose a reasonable initializer, e.g., the He initializer. You know, since the training set is too small, you may not even have to do batching.

   (b) Let's cheat here and use the *test set* of 10,000 examples as if they are our validation set. If you check on the test accuracy at every 100th epoch, you will see it overfit at some point. Record the accuracy values over iterations.

5. **The transfer learning task**:

   (a) Train our third classifier on the 500 CIFAR10 dataset you set aside in the begining. Again, note that they are for the original 10-class classification problem.

   (b) Instead of using an initializer, you will reload the weights from the pretext network. Yes, that's exactly the definition of transfer learning. But, because you learned it from an unlabeled set, and had to create a pretext task to do so, it falls in the category of *self-supervised learning*.

(c) Note that you can trasfer all the parameters in except for the final softmax layer, as the pretext task is only with 3 classes. Let's randomly initialize the last layer parameters with He.

(d) You need to reduce the learning rates for transfer learning in general. More importantly, for the ones you transfer in, they have to be substantially lower than 1×10^{-3}, e.g. $1 \times 10^{-5} or 1 \times 10^{-6}$. Meanwhile, the last softmax layer will prefer the default learning rate 1×10^{-3}, as it's randomly initialized.

(e) Report your test accuracy at every 100th epoch.

6. Draw two graphs from the two experiments, the baseline and the finetuning method, and compare the results. For your information, I ran both of them 10,000 epochs, and recorded the validation accuracy (actually, the test accuracy as I used the test set) at every 100th epoch. Of course, the point is that the self-supervised features should give improvement.