CSCI 4962/6962: Homework 6

Assigned Monday December 4 2023. Due by 11:59pm Friday December 8 2023.

Create a Jupyter notebook for this assignment, and use Python 3. Write documented, readable and clear code (e.g. use reasonable variable names). Submit this notebook making sure the answers to each question are legible and clearly labeled. You will be graded primarily based on the solutions and answers already present in the notebook, but it must also be runnable to reproduce your results. Do you remember the initial days of CAPTCHAs[1], when they consisted of images of sequences of letters and characters that you needed to convert to text? Nowadays CAPTCHAs have become torturous interactive and responsive multi-page affairs that tax your spatial and conceptual reasoning in addition to your eyesight. This is because the first generation of CAPTCHAs were exceedingly vulnerable to basic ML attacks. In this homework, you will somewhat prove this by using a CNN+RNN architecture for an OCR task very similar to CAPTCHA solving.

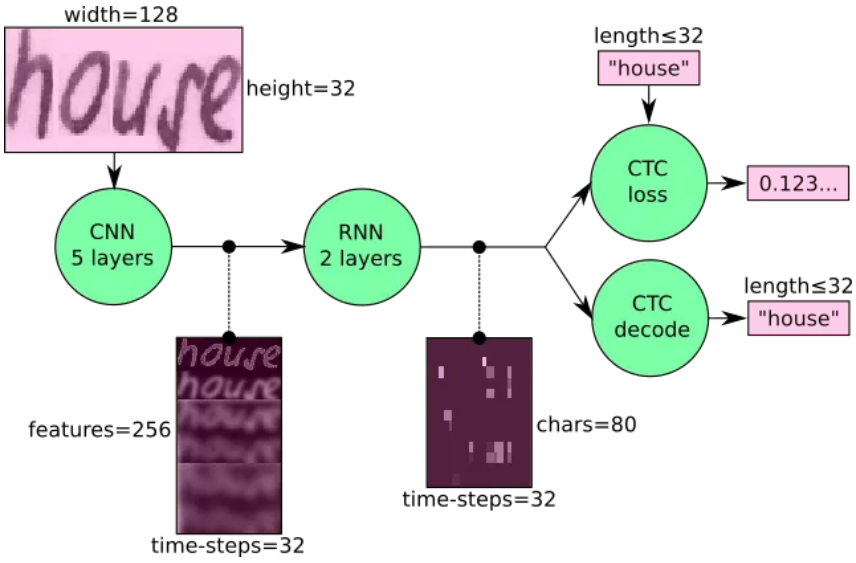

Figure 1: Overview of the architecture of your network. Taken from Harald Scheidl's article on handwritten text recognition.

NB, this will take time and compute power. Setup the environment on your laptop and get the model running properly there. You will train the model twice, for 300 epochs each, so you can infer the runtime by seeing how long one epoch takes. If the code runs too slow locally, try using GPUs through Google Colab. **I strongly recommend completing the code modifications necessary for the entire assignment and then running it for a small number of epochs locally first to check that everything– the display code included– runs end-to-end, before trying to run the full 600 epochs.**

1. Install the necessary packages. If you are not already using conda, I recommend using it with a virtual environment to prevent these changes from impacting your system-wide installation.

   Background The model you will build will take as input an image of variable width W and fixed height 32 as input. It will then use a convolutional architecture to convert this image to a feature matrix of size $W/4$-by-8. Each row should be viewed as containing information on a single character, so the network can be used to output a sequence of length $W/4$. The sequence of feature vectors are then fed through a (deep) bidirectional

[1] Completely Automated Public Turing tests to tell Computers and Humans Apart

RNN to convert them to $W/4$ softmax vectors, each of which represents an output character. Figure 1 gives an overview of the process.

Note that the input image may contain a sequence shorter than length $W/4$, as in the case of 'house', but the output sequence is always of length $W/4$. Further, if for example the 'h' in 'house' was particularly wide, the network may detect the letter 'h' in multiple consecutive positions. This is an example of one possible alignment issue between the ground truth sequence and the output sequence: multiple output sequences can correspond to the same input sequences. This and other alignment issues are standard in sequence modeling, and to handle them, we use the Connectionist Temporal Classification (CTC) loss.

To better understand the ideas behind the code you will be working with, read the Weights and Biases article on text recognition using a CRNN-CTC network.

torchvision Install torchvision. You will use it for data augmentation, specifically to translate, shear, and rotate the input images.

trdg Install the Text Recognition Data Generator (trdg). This will be used to generate the training data, consisting of pairs of images of short sequences of numbers, which will serve as input to our OCR model, and the corresponding sequences, which will serve as the labels.

Pillow You will use Pillow to read the input images. Be sure to install version 9.5.0 or earlier, as trdg is not compatible with the more recent versions. If you are using conda, `conda install ``pillow<=9.5.0''` suffices.

CTCDecoder You will train the OCR network using the CTC loss. Pytorch has a built-in CTCLoss function, but the CTCDecoder function in torchaudio did not work for me. Accordingly, install the CTCDecoder from Harald Scheidl's github

Batch generation functions Download a modified version of Avi Kutvonen's convenience code for generating minibatches of data using trdg. Keep this in the same directory as your Jupyter notebook.

Template code Download the template that you will complete to finish this assignment.

2. Complete the template by completing the definition of the `CNNBlock` module in the definition of the `OCR` module.

    1. Complete the definition of the `CNNBlock(c, k)` function, which takes as input the number of output channels (c) and the filter size (k). This function should return a pytorch `Sequential` module that consists of: 1) a convolutional layer with the specified number of ouput channels and the given filtr size, which does not change the height and width of the input image; followed by 2) a batch normalization layer (use `BatchNorm2d`); then 3) a `ReLU` activation. To avoid needing to specify the number of input channels, use the `LazyConv2d` module.
    2. Use your convenience function just defined to define the `CNNBlock` module as a `Sequential` module that comprises the following modules: `CNNBlock(8,3)`, `CNNBlock(16, 3)`, `MaxPool2d` over a 2-by-2 window, `CNNBlock(32, 3)`, `CNNBlock(64, 3)`, `MaxPool2d` over a 2-by-2 window.

3. Train an OCR model by executing the Jupyter notebook. Ensure that the plot of the validation loss and the table of example model outputs is visible in your submitted notebook.

5. Remove the RNN from the OCR model (flip the `useRNN` flag when calling the `OCR` constructor) and generate and show a similar convergence plot and example output table. Do this without overwriting your previous outputs. Answer these questions in a final cell in your notebook: how do the results change, qualitatively, and why do you think the RNN helps or does not help?