CMSC 25040 Computer Vision HW3 Writeup

Image Classification:

Data Preparation:

I split both the MNIST and CIFAR-10 datasets into two segments: training data and testing data. I additionally created a validation data split, although this happens in the train() function, and is discussed more in that section. I also increased the batch size to 50, as 4 seemed to be much too slow.

Model Structure / Optimizer:

- **MNIST Model:**
  My MNIST model structure was very simple. It included only a single convolution with kernel size = 3, stride = 1, and padding = 1 (one of the common convolutional layer configurations discussed in class). The layer outputs 8 channels. These outputs get batch normalized, then get passed through the ReLU activation function, and then get passed through a max pooling layer with kernel size = 2. Finally, the results get passed to a fully connected layer with 1568 inputs and 10 outputs, the number labels.

- **CIFAR-10 Model:**
  My CIFAR-10 model structure was a little bit more complicated. I have 4 convolutional layers in total, with each other convolutional layer doubling in size. Between every layer I apply the ReLU activation function, and between every second convolutional layer I apply batch normalization and max pooling with kernel size 2. After the convolutional layers I have three fully connected linear layers, the first of which has 4096 inputs and 1024 outputs, the second of which has 1024 inputs and 256 outputs, and the last of which has 256 inputs and 10 outputs, corresponding to the image labels. Between each layer there is a ReLU activation.

- **Optimizer:**
  For both the MNIST model and the CIFAR-10 model I used the Adam optimizer with the suggested parameters from class betas = (0.9, 0.999). One change I made was that I reduced the learning rate from the recommended 0.001 to 0.0005, as from a couple of initial tests it seemed like my model was learning too quickly and overshooting its optimal weights.

Training:

- For training, I additionally implemented early stopping and model saving. To implement early stopping, I first created a helper function validation_split_from_loader() which takes in the training loader and returns a new training loader and a validation loader. The validation loader loads items from a validation set, which corresponds to 10% of the original training set. At the end of each epoch, the current model's performance is tested on the validation set. If accuracy on the validation set has not increased in several epochs, then the model is likely overfitting to the training data. In this case, training is stopped and the model with the highest accuracy on the validation set has its parameters saved to model_name.pth using torch.save().

Testing

- For testing, the previously trained model is loaded from the model_name.pth location. Its performance on the test data is then evaluated, printing out the resulting accuracy.

Results

- **MNIST:**
    - **Training:** Early stopping around epoch 8, model from epoch 4 is used. Model has 97.95% accuracy on the validation set.

```
cuda:0
Epoch 0, Iteration 0, loss = 2.1835
Epoch 0, Iteration 100, loss = 0.4409
Epoch 0, Iteration 200, loss = 0.2257
Epoch 0, Iteration 300, loss = 0.4354
Epoch 0, Iteration 400, loss = 0.3453
Epoch 0, Iteration 500, loss = 0.2040
Epoch 0, Iteration 600, loss = 0.1599
Epoch 0, Iteration 700, loss = 0.2159
Epoch 0, Iteration 800, loss = 0.1399
Epoch 0, Iteration 900, loss = 0.0414
Epoch 0, Iteration 1000, loss = 0.0594
validation accuracy on validation set: 96.16666412353516
Epoch 1, Iteration 0, loss = 0.1561
Epoch 1, Iteration 100, loss = 0.1090
Epoch 1, Iteration 200, loss = 0.1599
Epoch 1, Iteration 300, loss = 0.2414
Epoch 1, Iteration 400, loss = 0.0914
Epoch 1, Iteration 500, loss = 0.1241
Epoch 1, Iteration 600, loss = 0.0424
Epoch 1, Iteration 700, loss = 0.0278
Epoch 1, Iteration 800, loss = 0.2441
Epoch 1, Iteration 900, loss = 0.0592
Epoch 1, Iteration 1000, loss = 0.0688
validation accuracy on validation set: 97.06666564941406
...
Epoch 8, Iteration 1000, loss = 0.0410
validation accuracy on validation set: 97.73332977294922
EARLY STOP DUE TO DECREASED VALIDATION ACCURACY: USING MODEL AT EPOCH 4
best model validation accuracy: 97.94999694824219
```

- **Testing:** 97.95% accuracy on test set

```
# Load model and test it on the testing set
test(mnist_test_data_loader, mnistNet1, "mnist_model")
```

```
Eval 9795 / 10000 correct (97.95)
```

- **CIFAR-10:**
  - **Training:** Early stopping at epoch 19, model from epoch 15 is used. Has an accuracy of 79.2% on the validation set

```
cuda:0
Epoch 0, Iteration 0, loss = 2.3082
Epoch 0, Iteration 100, loss = 1.4560
Epoch 0, Iteration 200, loss = 1.4030
Epoch 0, Iteration 300, loss = 1.5352
Epoch 0, Iteration 400, loss = 1.2495
Epoch 0, Iteration 500, loss = 0.7189
Epoch 0, Iteration 600, loss = 0.9608
Epoch 0, Iteration 700, loss = 1.0324
Epoch 0, Iteration 800, loss = 1.1941
validation accuracy on validation set: 64.22000122070312
Epoch 1, Iteration 0, loss = 0.9160
Epoch 1, Iteration 100, loss = 1.0343
Epoch 1, Iteration 200, loss = 0.6910
Epoch 1, Iteration 300, loss = 0.6907
Epoch 1, Iteration 400, loss = 0.9765
Epoch 1, Iteration 500, loss = 0.8560
Epoch 1, Iteration 600, loss = 1.0971
Epoch 1, Iteration 700, loss = 0.7546
Epoch 1, Iteration 800, loss = 0.6864
validation accuracy on validation set: 70.72000122070312
Epoch 2, Iteration 0, loss = 0.8453
Epoch 2, Iteration 100, loss = 0.5980
Epoch 2, Iteration 200, loss = 0.7450
Epoch 2, Iteration 300, loss = 0.4089
...
Epoch 19, Iteration 800, loss = 0.0111
validation accuracy on validation set: 77.23999786376953
EARLY STOP DUE TO DECREASED VALIDATION ACCURACY: USING MODEL AT EPOCH 15
best model validation accuracy: 79.25999450683594
```

  - **Testing:** 77.86% accuracy on test set

```
# Load model and test it on the testing set
test(cifar_test_data_loader, cifarNet1, "cifar_model")
```

```
Eval 7786 / 10000 correct (77.86)
```

Image Segmentation:

Visualization

To better understand the data I was working with, I created a small helper function visualize_samples which displays a random sample of images from the dataset. An example is shown below:

Model Structure / Optimizer:

- For my image segmentation neural network I implemented a neural network similar to UNet where intermediate features are saved during the encoding process, and are stacked on top of corresponding features during the decoding process. One difference between the architecture discussed in the UNet paper and my architecture is that I added dropout layers throughout the network in order to combat overfitting. The architecture is summarized as an encoding,decoding, and out processes, described as follows:
    - Encoding:
        - A convolution is applied to the input, doubling the number of channels (The first convolution goes from 3 to 64). The result is then passed through the ReLU activation function, and gets dropout regularization applied to it. A second convolution is applied to the result, keeping the number of channels constant. The result gets passed through the ReLU activation function again, and then undergoes max pooling.

        - The above process is repeated 4 more times, with the max pooling not getting applied at the last iteration.
    - Decoding:
        - An upconvolution is applied to the input halving the number of channels. The result gets dropout regularization applied to it. The result gets stacked together with the corresponding features from the encoding step and has a convolution applied to it, again halving the number of channels. The result gets passed through the ReLU activation function before getting convolved again, this time keeping the number of channels constant.
        - The above process is repeated 3 more times
    - Out:
        - Finally, we take the resulting output and apply a convolution with kernel size 1 and 20 output channels, which correspond to each of the 20 categories.
    - **Optimizer:**
      For the image segmentation model I again used the Adam optimizer with the suggested parameters from class betas = (0.9, 0.999). I again reduced the learning rate from the recommended 0.001 to 0.0005, as from a couple of initial tests it seemed like my model was learning too quickly and overshooting its optimal weights.

Training:

The train() function is the same as the train() function in the image classification section, with the small change that accuracy / loss is now measured by pixel, instead of by image.

Testing

The test() function is the same as the test() function in the image classification section, with the small change that accuracy / loss is now measured by pixel, instead of by image.

Results

- **Training:** Early stopping at epoch 58, model from epoch 52 is used. Has an accuracy of 74.08% on the validation set.

```
cuda:0
Epoch 0, Iteration 0, loss = 3.0027
Epoch 0, Iteration 100, loss = 2.4387
Epoch 0, Iteration 200, loss = 2.2617
Epoch 0, Iteration 300, loss = 2.2051
Epoch 0, Iteration 400, loss = 2.1924
Epoch 0, Iteration 500, loss = 2.1544
Epoch 0, Iteration 600, loss = 2.0745
Epoch 0, Iteration 700, loss = 2.0386
validation accuracy on validation set: 22.751543045043945
Epoch 1, Iteration 0, loss = 2.0640
Epoch 1, Iteration 100, loss = 1.9845
Epoch 1, Iteration 200, loss = 1.8809
Epoch 1, Iteration 300, loss = 1.9804
Epoch 1, Iteration 400, loss = 1.9037
Epoch 1, Iteration 500, loss = 2.0196
Epoch 1, Iteration 600, loss = 1.8547
Epoch 1, Iteration 700, loss = 1.8805
validation accuracy on validation set: 27.653614044189453
Epoch 2, Iteration 0, loss = 1.9179
Epoch 2, Iteration 100, loss = 1.8177
Epoch 2, Iteration 200, loss = 1.8116
Epoch 2, Iteration 300, loss = 1.8523
Epoch 2, Iteration 400, loss = 1.8335
Epoch 2, Iteration 500, loss = 1.8004
...
Epoch 58, Iteration 700, loss = 0.4288
validation accuracy on validation set: 73.95378875732422
EARLY STOP DUE TO DECREASED VALIDATION ACCURACY: USING MODEL AT EPOCH 52
best model validation accuracy: 74.07941436767578
```

- **Testing:** 73.93% accuracy on the test set.

```
test(loader_test, model, "seg_model")
```

```
Eval 7570501 / 10240000 correct (73.93)
```