Assignment 4 Part 2: Classification of ASL Images

Deadline: Thursday October 17, 2019 at 9:00pm

Late Penalty: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted.

The task in this assignment is to classify a set of images using Convolutional Neural Networks (CNNs), as a way to become familiar with CNNs. You will use the images that you and your classmates produced in Assignment 4 Part 1. Recall that you each generated three images for the ASL signs of the letters A-I, and K.

The learning goals are to becoming familiar with the basics of CNNs and their standard hyper-parameters, to learn how to debug neural networks, and to get exposure to some newer concepts including batch normalization and a widely-used loss function, called Cross Entropy Loss.

What To Submit

You should hand in the following files to this assignment on Quercus:

- A PDF file `assign4.pdf` containing your answers to the written questions in this assignment; please make it clear which sections contain the questions that you are answering.
- The complete code for this assignment, with just this one specification: all the models that you use (that are a subclass of the `torch.nn.Module` class) should be placed in a separate file called `model.py`. Your overall code have the ability to change all of the hyper-parameters as required in all sections of this assignment. Make sure there are enough comments are included in your code so that the TA grader can understand your approach.
- The trained model parameter files, `MyBest.pt` and `MyBestSmall.pt`, together with the hyperparameters used to create them, as described in Section 6.

1 Image Recognition using Convolutional Neural Networks

In this assignment you will code several Convolutional Neural Networks using the PyTorch library introduced in Assignment 3. PyTorch provides significant assistance in the design and coding of CNNs. There will not be any starter code provided in this lab, but you are free to use any code from previous labs, lectures and tutorials. You may use the PyTorch documentation freely. You might also find online tutorials helpful. However, all code that you submit must be your own.

Some of this assignment will be based on the PyTorch tutorial that shows how to build and train a convolutional neural net to recognize 10 classes from the CIFAR-10 data set. You may find it helpful to review this code: `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html`

2 Input Dataset [5 pts]

You can use the data that you personally created in Part 1 of this assignment to begin the creation of your code. When the full class dataset is ready, there will be an announcement and the data will be available under this assignment in Quercus.

The `torchvision` library is full of useful classes and methods needed in this assignment, and is described here: `https://pytorch.org/docs/stable/torchvision/index.html`. This includes holding famous datasets (such as the CIFAR-10 image dataset described above) that can be accessed and downloaded in a single line of code, as you can see in that example, as well as pre-trained models.

More relevant to this assignment, the `torchvision` library has two classes of direct use:

1. The `torchvision.transforms` class which makes it easy to do two important things with the images as they come in: First, you need to convert the image (which is a .jpg) file into a Tensor, so PyTorch can use it in its computations, using `tranforms.ToTensor`. Second, you need to *normalize* the image, making its values have an average of 0 and a standard deviation of 1, using `tranforms.Normalize`. This will require that you compute the mean and standard deviation of any of the data that you use this transform on. These should be combined into one composite transformation, using the class using `tranforms.Compose`, an illustration of which is given in the CIFAR-10 example, linked to above.

2. The `torchvision.datasets.ImageFolder` builds a the PyTorch `dataset` object (you used the `dataset` class in Assignment 3) which is in turn used by the `dataloader` object during training). The `ImageFolder` object builds the `dataset` from folders arranged on your computer. To use it, you simply point this class at the folder that contains (in this assignment) 10 other folders. Each of those folders represents a 'label' and should contain the images that correspond to that label, as described here: `https://pytorch.org/docs/stable/torchvision/datasets.html#imagefolder`. When you create a dataset for use in dataloading using this class, you can also have it invoke the composite transformation described above in point 1.

Set up the images that you captured for this assignment (just your images) in the directory structure used by `torchvision.datasets.ImageFolder`, and write the code to turn this into a `dataset` object and input it, properly transformed. Then, use the `dataloader` object to retrieve 4 images and display them.

Question: Print out and record the 4 images, along with their ground-truth labels, and put this into your submitted PDF file for this assignment. (You can see how this is done in the CIFAR-10 example, or you can make use of the dictionary that is provided in Dataset.ImageFolder.class_to_idx to see the labels). Check that the ground-truth labels are correct - if they are wrong, you have a bug that needs fixing!

3 Model Building and Testing [5 pts]

Code a basic CNN that has the following structure:

- two convolutional layers (using the relu activation function) followed by two fully-connected layers

- the first convolutional layer should use 4 kernels each of size 3x3, and a stride of 1
- the second convolutional layer should use 8 kernels of size 5x5 and a stride of 1
- each convolutional layer should be followed by a max pool layer of size 2 and stride 2
- the first fully-connected layer should use 100 neurons
- you should determine the number of neurons on the second fully connected layer

Write the training loop that takes in just your own images (**not** the set from the whole class and trains to the point of overfitting on your data. Use the following additional hyperparameters:

- Mean Squared Error Loss function (MSELoss)
- Stochastic Gradient Descent Optimizer (optim.SGD)
- Batch Size of 10
- learning rate of 0.01
- number of epochs 100 (but increase this if necessary to achieve perfect accuracy)
- be sure to initialize the torch random seed to a specific value, so that your code behaves the same way every time you execute it - using `torch.manual_seed(seed)`

One way to make sure that a neural network model and training code is working correctly, is to check whether the model is capable of "overfitting" or "memorizing" a small dataset. A properly constructed CNN with correct training code should be able to memorize the answers to a small number of images quickly. You should use this approach to debug all of your neural nets in the future, to make sure the input and output structure (i.e. all the code that feeds data into the neural net, and receives data from the net) is working.

Using just the images that you have collected, which is a very small dataset, show that your model and training code is capable of overfitting on this small data set. For this data, provide, in your report, the following (please note, you'll be asked to report these things for a number of experiments in other parts of this assignment):

- plots of loss and accuracy (of this training data only) vs. epoch
- the execution time of the full training loop, measured using the `time` library
- the number of epochs (determined by inspection) required to achieve 100% accuracy on the training data.
- the size of this network, in terms of: 1) The total number of parameters to be trained, and 2) the amount of computer memory required to store all of the parameters (weights and biases) in all layers of the network. To determine this, use the `torchsummary` library, which you will have to install, as described here: `https://github.com/sksq96/pytorch-summary`. (The README.md describes how to install the package, and to use it. You would import the `summary` function as shown and call in on your model as `summary(yourModel,3, 56,56)`. The output is self-explanatory, giving the counts of total parameters, and memory used.

With a large batch size (e.g. the entire small dataset) and learning rate that is not too high, you should be able to obtain a 100% training accuracy on that small dataset relatively quickly (within 200 iterations).

4 Using the Full Data Set [30 pts]

By Tuesday October 8th, the full dataset from the whole class should be available for this assignment on Quercus. Download that data.

4.1 Data Loading and Splitting [5 pt]

You must split the data into training and validation sets, which requires some careful thought. Recall that each person produces three images (samples) for each letter. There are two general ways to split between training and validation: the first would put some of everyone's samples into *both* the training and validation set. The second would put only some people's images into the training set and others only into the validation set.
Questions:

1. Which method is better? Why?
2. What fraction of train/validation split did you use? Why?
3. Report the mean and standard deviation of the image *training* data that you use.

4.2 HyperParameters

In this assignment there are quite a large number of hyperparameters to explore. These include:

1. Number of Convolutional Layers
2. Number of kernels on each convolutional layer
3. Size of the kernels on each convolutional layer (use only 1 size)
4. Activation function
5. Number of fully connected layers
6. Number of Neurons on all but the final fully-connected layer
7. Learning rate
8. Batch size
9. Loss function
10. Optimizer
11. Use of batch normalization

4.3 Hyperparameter Search 25 pt

First, fix the following hyper-parameters to be as follows:

- Activation Function: ReLU
- Optimizer: optim.SGD
- Batch Normalization: no

- Loss Function: MSELoss
- Size of the kernels on each convolutional layer 3x3
- Number of Fully Connected Layers: 2

For the remaining parameters, consider the following ranges:

1. Number of Convolutional Layers 1,2,4
2. Number of kernels on each convolutional layer 10, 30
3. Number of Neurons on first layer: 8, 32
4. Learning rate .1, .01
5. Batch size 4, 32

If you were to try all possible combinations, that would be 48 separate experiments, which is too many. Instead, select a subset of 12 combinations, which you think are best to train on.

Questions: Indicate which 12 combinations you used, and explain why you chose them. Present the following results for each of the combinations:

- validation accuracy
- training execution time
- number of epochs to train (i.e. how many epochs until the training is no longer improving the validation answer)
- number of parameters in network (see Section 3)
- amount of memory space required to hold the parameters in network (see Section 3)

Once you have finished this exploration, comment on the relative merits of different networks with respect to these metrics.

For the network that you select as best, provide the loss and accuracy vs. epoch plots, for both the training data and validation data.

5 Batch Normalization and Cross Entropy Loss Function [5 pt]

In class we have described Batch Normalization and Cross Entropy Loss functions. For the best network that you found in the previous section, create three new versions:

1. One that adds (to the best network) the use of batch normalization; it should be used in both the convolutional layers and fully-connected layers.
2. One that, for the best network, replaces MSEloss with Cross Entropy Loss.
3. One that uses both batch normalization and cross Entropy loss

Give all of the same metrics reported above in a table, and provide both the loss and accuracy plots for training and validation. Discuss the effect of these two changes on the results.

6 Final Best Networks and Confusion Matrix [10pt]

First, explore and find the best (most accurate) network that you can using *any* other network that you train from scratch, with no restrictions on the above hyper-parameters. (One rule: you may not use a pre-trained network of any kind). Store the trained parameters of your most accurate network in a file using the `torch.save(model.state_dict(),'MyBest.pt')` Pytorch method. You should submit this file as one of the files you upload to Quercus. Indicate which model from the submitted code file `model.py` was used, and provide all of the hyper-parameters settings you used in its creation.

Second, find the most accurate network that uses less than 5000 trainable parameters, under the same rules as above. The number of parameters should be measured by the `torchsummary` library `summary` function. Include in your report the full output from the summary function for the network. Provide the parameter model for this network in the file `MyBestSmall.pt`. Indicate which model from the submitted code file `model.py` was used, and provide all of the hyper-parameters settings you used in its creation.

Questions: For your most accurate network (#1 above), provide the *confusion matrix*, which is a 10 x 10 matrix that shows, for each of the 10 possible ground truth answers, how many predictions in each of the 10 classes occurred across for the validation set. You should use the `sklearn` library function `confusion_matrix` which you can see described here: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html`

Looking at the confusion matrix, which two classes were the most *confused* for your model? Why?

There will be two prizes, one for the best accuracy in each of the two cases above.