**Revanth Tiruveedhi**

**Problem 2. Understanding Performance Tunning for Training High Accuracy Deep or Shallow Models**

In this problem, you are given 2 options. In each option, you are asked to train multiple models using different hyperparameter configurations (option 1) or using different dataset complexity plus hyperparameter configurations (option 2).

**Problem 1.1 Performance Tunning for Deep or Shallow Learning**

You are asked to train 5 models for the same learning task and dataset, each uses one configuration of the hyperparameters: the default configuration from the package you download, and the four different configurations you are asked to change and each of the four should have only one hyperparameter different to the baseline to help with the analysis on performance tuning results. If you use CPU without GPU card, then create a smaller dataset. Here are the steps:

1. Choose your favorite ML framework, say TensorFlow, Scikit Learn, PyTorch, etc.. In the rest of the requirement, we will use TensorFlow as an example.

**I have chosen to use Pytorch on an Anaconda framework over Jupyter Notebook environment.**

1. After download TensorFlow at <https://www.tensorflow.org>. Following the instruction to install it on your laptop or desktop computer: <https://www.tensorflow.org/install/install_linux#ValidateYourInstallation>. There​ ​are​ ​several ​options​ ​to​ ​install​ ​TensorFlow​ ​on machines​ ​without​ ​GPU​ ​card(s). For example, installing​ ​Tensorflow​ ​on​ ​virtualenv​ ​will​ ​isolate​ ​your​ ​Tensorflow’s​ ​python​ ​environment.

**Installing Anaconda Navigator that includes Jupyter, I installed some dependencies such as scikit learn, matplotlib and numpy to help with computations**

**Graphical user interface

Description automatically generated with low confidence**

1. Choose a dataset and a machine learning (ML) algorithm to train a baseline ML model using the default hyperparameters, say CNN3 MNIST, which use a 3 layer CNN to train a 10-class neural network classifier on MNIST dataset. Given MNIST dataset has 60K images. You may choose a subset of them, say 6000 total, evenly distributed for 10 classes, say 600 per class.

**My model uses 10 epochs with 60 train items per cycle of the epoch for a total of 600 train images per epoch. The model is a 3 layer CNN that trains a 10 class neural network on the MNIST dataset.**

1. You are asked to show the default settings of the hyperparameters of the TensorFlow CNN algorithm you used to train a k-class classification model, such as the CNN layers used, the #kernels (weight filters) used, the feature map sizes, the minibatch size, the #iterations (#epochs), the loss optimizer (e.g., SGD, Adam, etc.)

Kernel – 3x3

Minibatchsize – 60

Iterations – 10

Loss optimizer – Adam

Graphical user interface, text, application, email

Description automatically generated

1. Train your model using the algorithm and the dataset you have selected. Report the output of the trained K-class model, including

* the storage size of the model in MB, -
* the test accuracy (use a small portion of the training data as training validation)

Chart

Description automatically generated

Chart, line chart

Description automatically generated

* the training time

Graphical user interface, text, application

Description automatically generated

* the test time.

A picture containing text

Description automatically generated

* In addition, you may want to show the confusion matrix for all K classes. For K>10, you may choose to show a selected subset of classes. [Hint: see lecture 6 on confusion matrix] – This has a class of 10, and not greater than 10

1. **Outlier Test Scenario**. When performing testing on a k-class classifier you have trained, in addition to use the test dataset from the same collection, you are asked to perform an outlier test by using 10 or more images that do not belong to the dataset of your learning task. For example, if you choose dog and cat as your binary classification task, then create 10 photos of your favorite actor/actress, or favorite cars which will be the out of distribution data. Perform outlier test on your classifiers and report your results in a table titled outlier test. Also include 5 examples of your outlier test set in your report.
2. For each of the 10 outlier queries, and run each of the outlier tests at least three times and report your results, including
   * The results from the 3 times of the same outlier query: show if they return the same prediction result or not;
   * Make an attempt to elaborate on why the results from 3 times of test using the same test example may result in very different probability vectors.

**Hint:** There are two ways to perform the above tests: (i) select 3 different pre-trained models on the same dataset, run each of the outlier tests against three models and analyze your results on the above two bullets. (ii) perform the same outlier query against the same model. Report your finding. Ideally compare with case (i). Elaborate your comparison results and your thought on the results.

1. Now choose at least two different types of hyperparameters (say different learning rate functions and different #kernel filers) and for each type, select two settings (say fixed learning rate with two different values, plus 10 kernel filers v.s. 5 kernel filers) and use the same training dataset (say MNIST) to train 4 different models for the same learning task (e.g., MNIST with 10 classes), each using one changed hyperparameter configuration, and compare the performance of these 4 newly trained models with the baseline model you have trained in step (1)-(7) in

* training time,
* test accuracy,
* test time,
* the storage size of the trained model

1. Analyze the comparison results based on the confusion matrix and the different hyperparameter configurations you have used in the 5 models you have trained.

Hints: Hyperparameters include (i) different #kernels (weight filters) used, (ii) different learning rate function or settings, say different value for a constant learning rate; (iii) different convergence condition, e.g., #iterations (or #epoches), (iv) other hyperparameters of your own choice, including different training algorithm: say you used LeNet MNIST and now change to another algorithm, such as ResNet32, ResNet64, MobiNet, VGG16, …

**Deliverable:**

1. provide URL of your open source code package and the dataset download.

Provided on link to Jupyter notebook:

Open Source Help: <https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118>

Pytorch: *conda install pytorch torchvision torchaudio cpuonly -c pytorch*

Dataset:

*part\_tr = datasets.MNIST(*

*root = 'data',*

*train = True,*

*transform = ToTensor(),*

*download = True,*

*)*

*part\_te = datasets.MNIST(*

*root = 'data',*

*train = False,*

*transform = ToTensor()*

*)*

1. Screen shots of your execution process/environments

Jupyter Notebook is on file in the ZIP

Graphical user interface, text, application, email

Description automatically generated

1. **Input Analysis**: *Use a table to report your training configuration parameters*:
   1. the input dataset (size, resolution, storage size in KB or MB per image, storage size of dataset in MB or GB).

Each MNIST image is 28x28 grayscale image with one feature per pixel (ranging from 0-255). The total storage size being ~50 MB

* 1. choose and show 5 sample images per class for all K classes in the dataset.

0:

    

1:

    

2:



3:



4:



5:



6:



7:



8:



9:



* 1. the training v.s. testing data split ratio and size used in your CNN training.

6000 train and 1000 test (85% and 15%)

* 1. You are asked to record the structure and default settings of the neural network (NN) algorithm you chose to train your k-class classifier, such as LeNet, or ResNet, or DenseNet, and the default NN structures (e.g., CNN with at least 2~5 convolutional layers), and the default hyper-parameters, such as neuron size, the number of weight filters (kernels) and the size of kernel (weight filter) for each layer of your CNN, the min-batch size, #epochs/#iterations (convergence), in a table for the configuration you use to train your four CNN classifiers.

Sequential, ReLU(), MaxPool2d

* 1. You are asked to report the default #iterations used for training convergence.

The default iterations are 6000 iterations

1. **Output Analysis:** *Report the performance comparison and analysis of each of your 5 CNN classifiers:*
2. You are asked to provide a table to compare the 5 models you have trained in terms of training time, training accuracy, testing time and testing accuracy.
3. You are asked to record the trained model size in MB in the above table for all 5 models.
4. You are asked to make at least three observations from your experimental comparison of the 5 models that you have trained on the same dataset under 5 different hyperparameter configurations (including the baseline model using the default hyperparameters).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Training Time | Training Accuracy | Testing Time | Testing Accuracy | Model Size |
| Base | 619 | 0.99 | 5.78 | 0.99 | 1.3 |
| Kernel 5 Adam | 1154 | 0.98 | 4.41 | 0.96 | 2.4 |
| Kernel 5 SDG | 875 | 0.98 | 4.39 | 0.97 | 1.4 |
| Kernel 3 Adam | 726 | 0.99 | 4.165 | 0.99 | 1.3 |
| Kernel 3 SDG | 714 | 0.97 | 4.09 | 0.95 | 1.2 |

There are two big changes that happened in this experiment, one being that there is a change of 5 kernels (filter sizes) and the other being the effect of the optimizer function. With the base comparison between the other experimental values, we see that there are multiple factors with why kernel would impact the performance and the overall attributed accuracy. With the training time of the Kernal 3, we see that there is a significantly less amount of time that it takes as there are less of an output since the fully connected layers will be 32\*5\*5 while the the filter of size 5 will have a 32\*7\*7 layers total. In addition, we see that there is a performance decrease with a higher kernel, and doing research, I found that with a higher kernel there is potential for overfitting and estimation given that we have a 28 x 28 pixel image. Finally with the comparison of the optimizers, SGS is a variant of gradient decent and it performs it on a random selection of examples, while Adam is an algorithm that does gradient optimization. For this reason we can show that the accuracy between the optimizers can related to why Adam is higher that SDG

**Experiment:**

**Experiment 1**

**Text, letter

Description automatically generated**

**Text

Description automatically generated with medium confidence**

**Chart, line chart, histogram

Description automatically generated**

**A picture containing text

Description automatically generated**

**Experiment 2**

**Text, letter

Description automatically generated**

**Text

Description automatically generated**

**Graphical user interface, application

Description automatically generated**

**A picture containing text

Description automatically generated**

**Experiment 3**

**Text, letter

Description automatically generated** **Text

Description automatically generated**

**Graphical user interface, chart, line chart, histogram

Description automatically generated**

**A picture containing chart

Description automatically generated**

**Experiment 4**

**Text, letter

Description automatically generated**

**Text

Description automatically generated**

**Graphical user interface, application

Description automatically generated**

**A picture containing text

Description automatically generated**