Facial Recognition using Neural Networks Developed from Scratch

Chunnan Sheng
Cong Hai Nguyen

1. Introduction

This project is originated from the Carnegie Mellon University assignment 20 years ago. Purpose of the original assignment was to experiment facial recognition using neural network toolkit programmed by C. Instead of using the C program, we use a new C++ library developed by Chunnan Sheng to add in functions such as convolutional neural network (programmed from scratch), stochastic mini-batch learning and multi-threading to expand our understanding of facial recognition that is not available through using the original C program.

To extend the basic test cases that were provided by the original assignment, we tried Transfer Learning method to explore potentials in CNN (convolutional Neural network) and DNN (fully connected network). Throughout the project, we experienced several challenges; The two major problems we overcame were transferring the source code from Windows platform to Linux platform and overcoming memory limitation problem when testing the program on remote Linux machine via SSH connection.

2. Implementations

We have recreated all the test scenarios using python scripts based on C++ programming language instead of the original source code provided because there are many deprecated functions that are out of date.

The C++ library was originally created by Chunnan Sheng. It is extended to enable the library to suit this project. Deep learning framework such as TensorFlow, Caffe and Keras are not being used in this project.

The purpose of the original C++ library was to implement multi-layered fully connected neural network (dnn) and convolutional neural network (cnn). The original implementations of the C++ library are as follows.

1. Matrix calculations
2. Convolutional calculations of matrices
3. Activation functions and error functions
4. Back propagation mechanism
5. Max pooling method
6. Linear Regression using gradient descent (Unrelated to this project)
7. One Dimensional GMM EM algorithm (Unrelated to this project)
8. Support of MNIST and CIFAR_10 datasets (Unrelated to this project)

New functions and mechanism listed below are added to the library to accomplish the needs of the project

1. Random mini-batch learning (size of mini-batch is configurable)
2. Multi-threading for mini-batch learning (number of threads is configurable)
3. Saving of trained DNN and CNN status (weights) to computer storage so that the neural network can resume its training after it is stopped.
4. Support of facial dataset from CMU. The CMU dataset includes 1888 facial images of PGM file format.
5. Momentum mechanism for gradient descent (momentum is configurable within [0, 1])
6. Python scripts to run test cases derived from the original CMU assignment.

Diagram 1 and 2 illustrate overview of source code structure. There are generally 4 modules implemented: neurons module, dataset module, cnn module, dnn module and facetrain module.

Diagram 1 Structure of neurons module

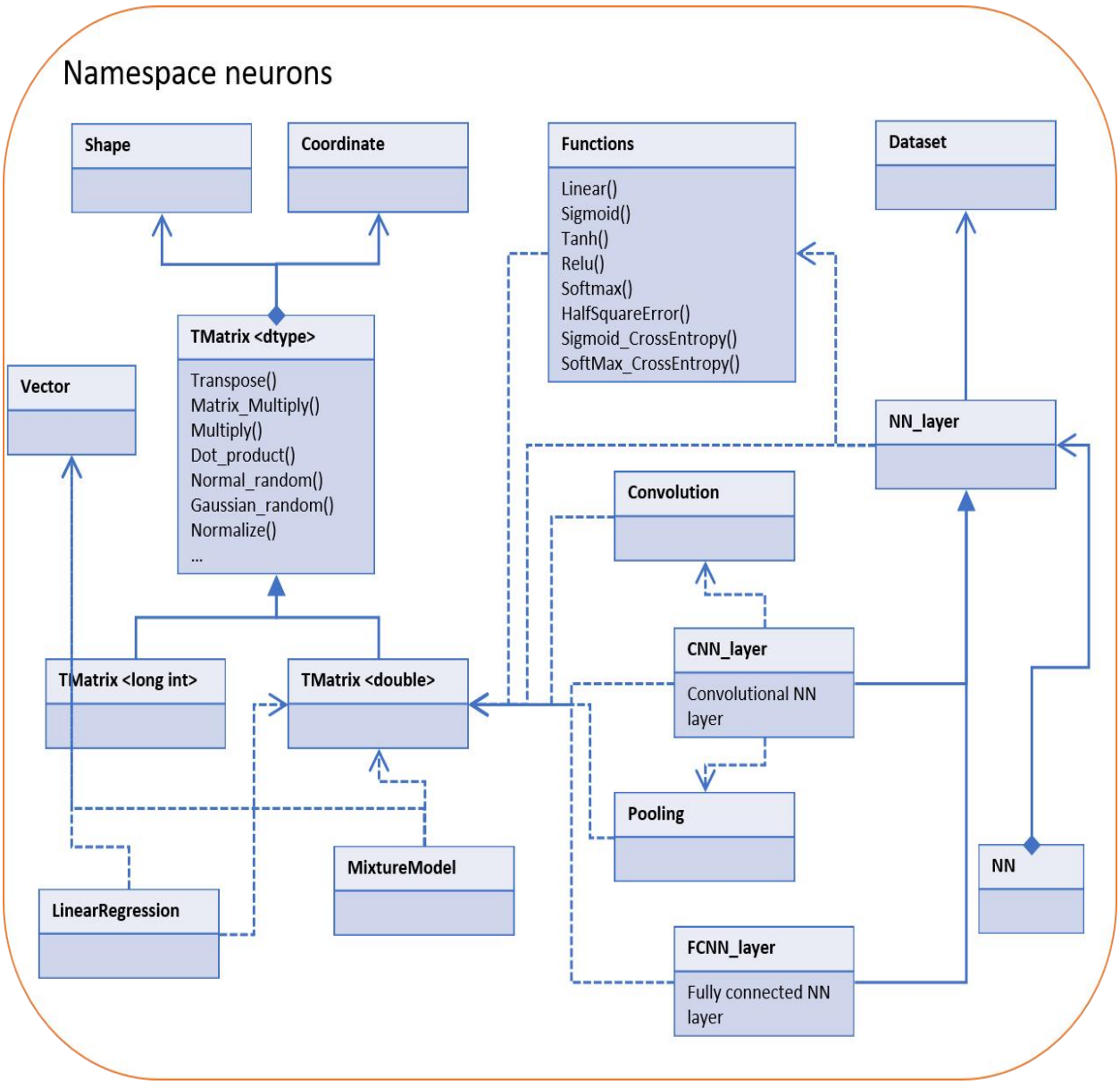


Diagram 2 Relations between neurons module and other modules

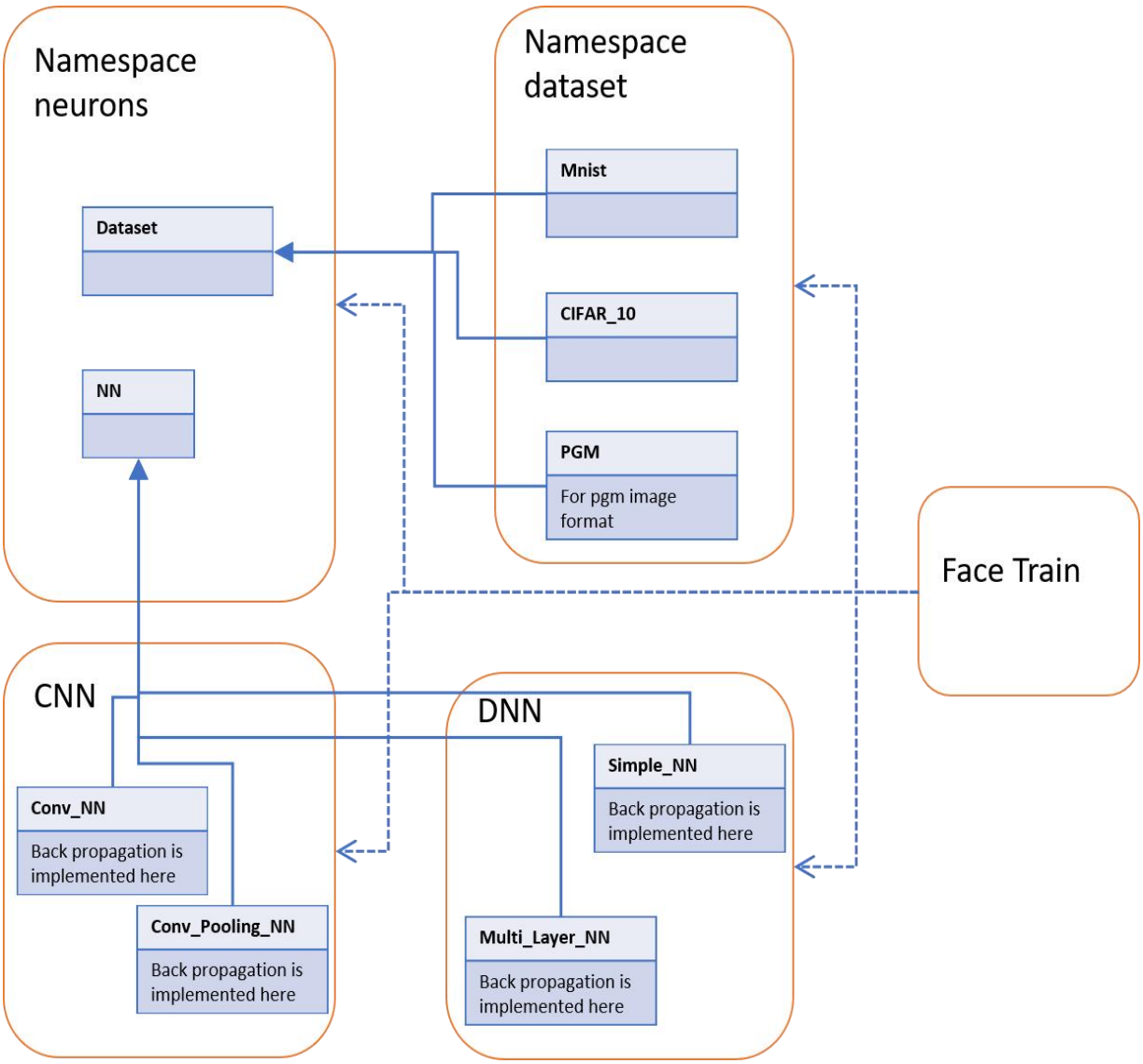
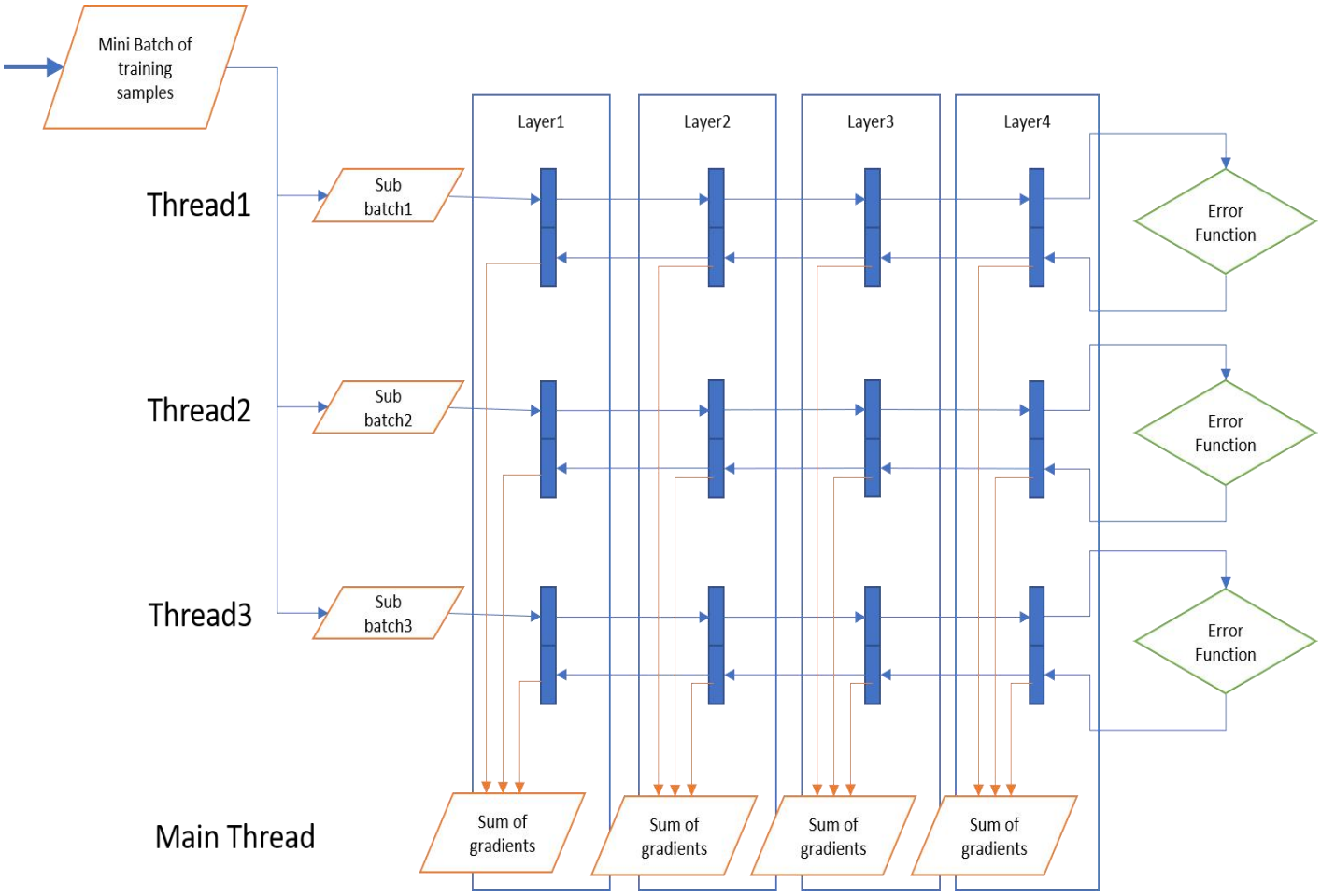


Diagram 3 shows how backpropagation, mini-batch learning, and multi-threading are implemented.

Diagram 3 Graphical illustration of mini-batch learning and multi-threading



3. Basic Experiments

A. General description

There are 4 individual tasks to be performed for each image: recognizing sunglasses, recognizing a person's face, recognizing a person's expression (neutral, angry, happy, sad), and recognizing the face's position (left,right, or up). Two type of neural network will perform each of these 4 tasks: deep neural network and convolutional neural network.

B. Setup

Uncompress "files.tgz" and go into the resulting "projects" folder.

C. Running automated tests:

To test results from varying the parameters of the network, we have created automated test script called "test.sh" in the folder "projects/facetrain/tests". When executed, it will automatically run the program with various settings and stored both the parameters as well as the results in the corresponding output files for later analysis. The general syntax for running our automated tests is: "bash test.sh -t <test_param> <path_to_test_file> <network_type>". <network_type> is always either "dnn" or "cnn", and the meaning of the arguments are:

To test **a particular category**: <test_param> is one value of epoch, momentum, or nthreads, and <path_to_test_file> is not used (i.e. leave it out of the command line).

To test a **particular test file** in a particular folder: <test_param> is epoch, momentum or nthreads, and <path_to_test_file> is the path to the filename

When a new test run, it will overwrite the file of same test run previously.

D. Running the program manually

User can run the program manually with the following syntax:
From the directory projects/facetrain/bin/x64/Release/, enter the following command line:

./facetrain.out [-N <network_type>] -n <network-filename> [-f <network to borrow from>] [-t <train-filename>] [-1 <test1-filename>] [-2 <test2-filename>] [-E <epoch size>] [-e <number of epochs>] [-s <seed>] [-S <number of epochs between network saves>] [-l <learning-rate>] [-a <nthreads>] [-m <momentum>] [-b <batch-size>] [-H] [-T]

In the syntax above, the square brackets indicate optional value. The meanings of the parameters are:

- E: epoch size (default 10)
- N: type of network used (“dnn”, “cnn”, etc)
- S: epoch counts between two adjacent network saves (default 100)
- H: if this flag is set, the network’s weights will be saved to file (default false)
- a: number of threads (default 4)
- m: momentum (default 0.3)
- b: batch size (default 8)
- e: number of epochs (default 100)
- f: filename of network to borrow from
- l: learning rate (default 0.001)
- n: filename for saving network states
- s: seed (default 1.0)
- t: filename containing a list of training files
- 1: filename containing a list of files for the first test
- 2: filename containing a list of files for the second test

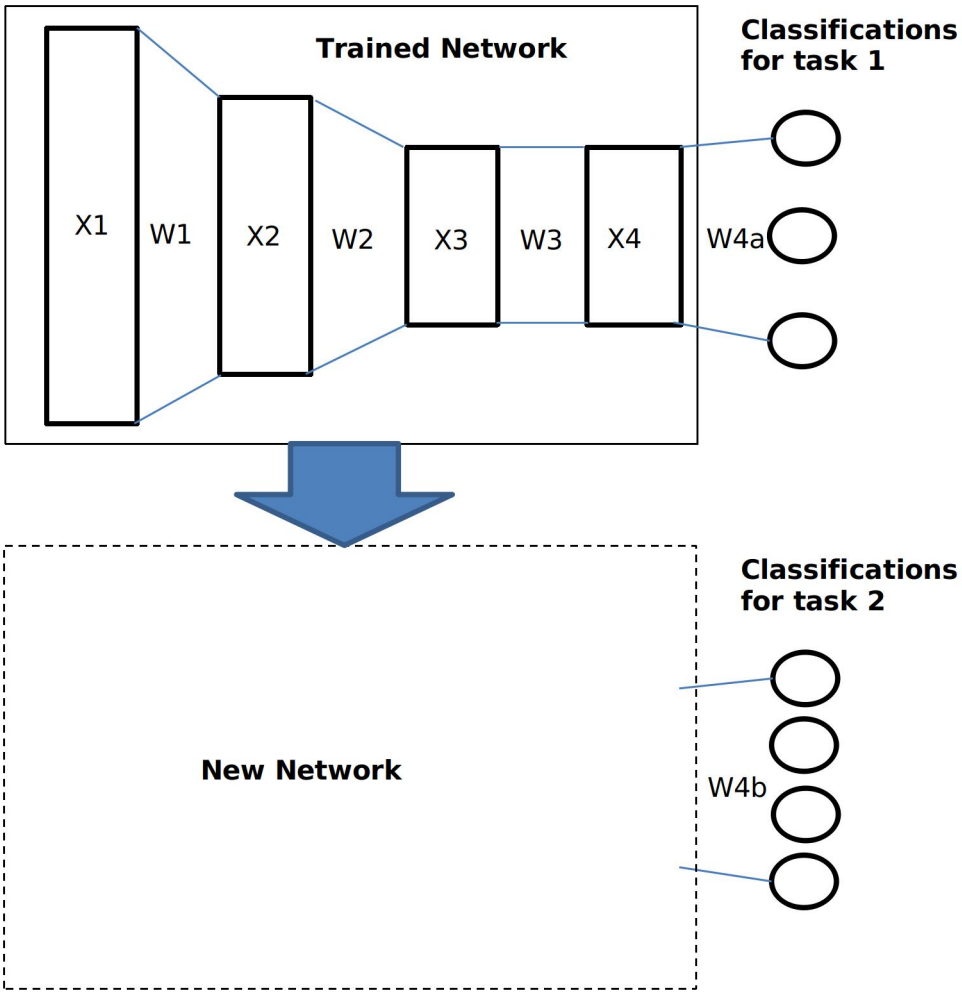
E. Results

The maximum classification accuracy achieved is 99%. The machine performance is best on recognizing face, sunglasses, and pose, while it had a hard time with recognizing expression. “dnn” network not only has higher accuracy than “cnn” in general, it also runs faster than the latter.
Results also show that the number of epochs, the value of momentum, and the number of threads has a positive correlation to accuracy (please find the Excel data file attached).

4. Experiment of Transfer Learning

We tried Transfer Learning using a trained network for pose classification to train another network for face classification. In this experiment, ***-f*** command option is introduced here to “borrow” a trained network to train another network. For example, ***./facetrain.out -n face -f pose*** means a network named pose is borrowed to train another network named face.

The following picture shows how Transfer Learning experiment is done in this project. The entire network except the last layer is copied to the new network.



A comparison experiment is done using CNN (convolutional neural network) and DNN (fully connected neural network).

What we see from this experiment is that Transfer Learning can increase training accuracy of CNN. In contrast, Transfer Learning fails to improve performance of DNN and even negatively affect the training accuracy.

Transfer Learning experiment on CNN

Step 1: Train a network for pose classification (400 epochs)
./facetrain.out -n pose -t all_train.list -1 all_test1.list -2 all_test2.list -e 400 -s -1 -S 40 -N cnn -b 8

Step 2: Train face classification network with transfer learning using pose network (40 epochs)
./facetrain.out -n face -f pose -t all_train.list -1 all_test1.list -2 all_test2.list -e 40 -s -1 -S 40 -N cnn -b 8

Step 3: delete the face classification network

rm ./cnn_face

Step 4: Train face classification network without transfer learning (40 epochs)

./facetrain.out -n face -t all_train.list -1 all_test1.list -2 all_test2.list -e 40 -s -1 -S 40 -N cnn -b 8

Step 5: repeat step 3

Repeat 4 times step 2 to step 5 several times to get similar results from the below table (due to randomization, the numbers may vary slightly)

	Training accuracy with transfer learning after 40 epochs	Training accuracy without transfer learning after 40 epochs
Trial 1	0.576923	0.221154
Trial 2	0.552885	0.173077
Trial 3	0.576923	0.197115
Trial 4	0.591346	0.192308

Clearly the table shows that there is a significant increase in training accuracy when using transfer learning in the experiment.

Transfer Learning experiment on DNN

Step 1: Train a network for pose classification (50 epochs)
./facetrain.out -n pose -t all_train.list -1 all_test1.list -2 all_test2.list -e 50 -s -1 -S 50 -N dnn -b 8

Step 2: Train face classification network with transfer learning using pose network (5 epochs)
./facetrain.out -n face -f pose -t all_train.list -1 all_test1.list -2 all_test2.list -e 5 -s -1 -S 5 -N dnn -b 8

Step 3: delete the face classification network

rm ./dnn_face

Step 4: Train face classification network without transfer learning (5 epochs)
./facetrain.out -n face -t all_train.list -1 all_test1.list -2 all_test2.list -e 5 -s -1 -S 5 -N dnn -b 8

Step 5: repeat step 3

Repeat 4 times from step 2 to step 5 to get a similar results from the below table (due to randomization numbers may vary slightly)

	Training accuracy with transfer learning after 5 epochs	Training accuracy without transfer learning after 5 epochs
Trial 1	0.418269	0.610577
Trial 2	0.322115	0.615385
Trial 3	0.302885	0.538462
Trial 4	0.336538	0.6875

Clearly the table shown that there is a significant decrease in training accuracy when using transfer learning in the experiment.

5. Appendix

- The code design
- Compiling from source
    - Requirements:
        - Windows 10
        - Visual Studio 2017 with “C++ project” and “Cross-platform - Linux” packages installed
    - Set up:

While in Visual Studio, open the file “my_neurons\neurons_linux\neurons_linux.sln”
From Visual Studio menu: select Build -> Configuration Manager -> “Active Solution Platform:” x64 -> “Close”
From Solution Explorer window, for each of the following directory, manually right click then select “Build”: “cnn” -> “dataset” -> “dnn” -> “neurons” -> “facetrain” (please follow this build order)
Visual Studio may ask for the remote target machine. If so, please enter your login details for cse lab machine.
After the build completes, the executables can be found on the cse lab machine at folder ~/projects/facetrain/bin/x64/Debug/facetrain.out, as well as on local project folder at my_neurons\neurons_linux\facetrain\bin\x64\Debug\facetrain.out

6. References

K. Yamanishi. 1998. *A decision-theoretic extension of stochastic complexity and its applications to learning*. https://ieeexplore.ieee.org/abstract/document/681319/.

R Hecht-Nielsen. 1992. *Theory of the Backpropagation Neural Network*. https://www.sciencedirect.com/science/article/pii/B9780127412528500108.

S. Lawrence; C.L. Giles; Ah Chung Tsoi; A.D. Back. 1997. *Face recognition: a convolutional neural-network approach*. https://ieeexplore.ieee.org/abstract/document/554195/.