Image Classification with MNIST

In this tutorial, you will apply image classification using Convolutional Neural Networks(CNNs), trained on the DBC network, to recognize images of digits. The dataset, MNIST, is a common entrypoint dataset to get started with deep learning.

Prerequisites

- Knowledge of basic Linux command line instructions
- Basic knowledge of supervised learning and CNNs
- Created an account on the dbc website with sufficient dbc for renting a GPU
- Keras > v2 installation on host machine / laptop
- Access to DBC AI Manual

Approximate time for completion: 1 hour

Download the required code

Download the repo for this tutorial: www.github.com/dbc_mnist

Loading data

Run ‘download_mnist.py’ on your local machine. This will download the MNIST dataset in a folder ‘data_dir’. It consists of a training set of 60,000 images, and a test set of 10,000 images (see fig 1 for examples).

```
python3 download_mnist.py
```

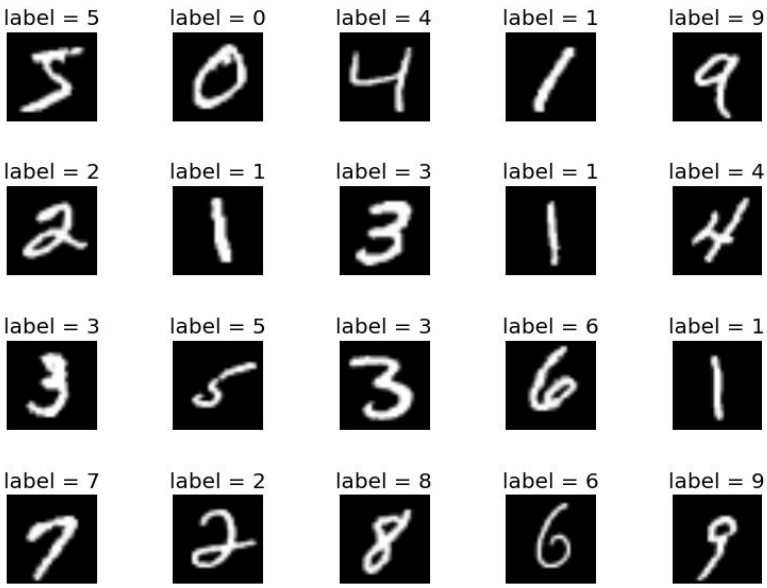

Fig 1 - The MNIST dataset

Next, upload the data directory to DBC:

```
cd your_dbc_installation_directory/dbc_repo/tool/
bash ./dbc_upload /home/user/path_to/data_dir/
```

You will see a similar display on your terminal:

```
dbc_upload | input /home/user/path_to/data_dir/
dbc_upload | add to local repo
dbc_upload | publish content
dbc_upload | upload content to
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100 52.4M    0 52.4M    0     0   223k      0 --:--:--  0:04:00 --:--:--
144k
DIR_HASH:QmNtgaaaf52bcacw3aaa2EzBgf9MkFHxaaaaaeu8xdQa
dbc_upload | content upload completed
```

Copy the DIR_HASH (the one highlighted in red is only an example!).

Paste the DIR_HASH in the task.config file in the downloaded code directory. It should look similar to the following:

```
select_mode=0
peer_nodes_list=

#GPU
training_engine=kbobrowski/tensorflow-gpu-opencv

data_dir=QmNtgaaaf52bcacw3aaa2EzBgf9MkFHxaaaaaeu8xdQa
code_dir=
entry_file=run.sh
checkpoint_dir=
hyper_parameters=
```

The training file

The mnist.py script is the main training file for this tutorial. The steps it covered are the following:

1) Preprocess the data

We first reshape the data to have specific dimensions required by keras. MNIST images have a height and width of 28 pixels and a depth of 1 (grayscale or 'black and white' images have a depth of 1 in general).
Next, the input data type is converted to float32 and normalized to the range [0, 1].

```
X_train = X_train.reshape(X_train.shape[0],28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

Now we convert the labels such that each label is represented as 10 distinct class labels, instead of the actual class value.

For example,
The label 0 will be represented as [1000000000]
The label 3 will be represented as [0001000000]
The label 8 will be represented as [0000000010]

This process is called one-hot encoding.

```
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
```

2) Define our model

We first create a Keras Sequential object.
Next we add convolutional layers, followed by a max pooling layer, and a dropout.

The weights from the Convolution layers must be flattened (made 1-dimensional) before passing them to the fully connected Dense layer.

We then add our fully connected (dense) layer and output layer to complete the model architecture.

```
model = Sequential() # create sequential object

model.add(Convolution2D(32, 3, 3, activation='relu',input_shape=(28,28,1)))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

3) Compile our model

We must compile out model before we start the training process. In this step we define some useful parameters such as the loss function, and the optimizer.

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

4) Train the model

We will now begin the training process on the training data and labels. The number of epochs has been set to 10. The means that the entire training data will be 'looped over' 10 times. This number can easily exceed 100 for larger datasets.
The batch_size refers to the size of the set of data entries that are used for each training step. There are multiple batches in a single epoch (a total of epoch/batch_size training steps). We usually set the batch size based on the available memory that is available in out computing devices. All the GPUs in the DBC network can handle the default value.

```
model.fit(X_train, Y_train,
          batch_size=32, nb_epoch=10, verbose=1)
```

5) Test the model

We finally evaluate the trained model on the test set of data. The score will contain the accuracy value of the evaluation process, to gauge the effectiveness of our model.

```
score = model.evaluate(X_test, Y_test, verbose=0)
```

Loading the code directory to DBC

Upload the code directory to DBC:

```
cd your_dbc_installation_directory/dbc_repo/tool/
bash ./dbc_upload /home/user/path_to/data_dir/
```

As was done for the data directory, copy the DIR_HASH and paste it in the task.config file for the `code_dir=` field.

Using DBC for Training

We will be using the DBC platform to access a GPU for training.

Login to the DBC website and purchase 1 GPU for 1 hour, which is the minimum duration for GPU rentals.

Next, note the machine hash as instructed in the AI Manual and paste it in the `peer_nodes_list=` of the task.config file

Now that the task file is complete, start the training process by executing the task. In your dbc terminal, run the following:

```
dbc>>> start -c /home/user/path_to/task.conf
```

Note the task id:

```
task id: qT1fzqHLcLEiCdcetLRQcWq5zMDJakyaPasdJrVi7xvr5sxeM
create_time: 09/13/18 15:09:22
```

You can check the status of the task from time to time using the following command:

```
dbc>>> logs -t qT1fzqHLcLEiCdcetLRQcWq5zMDJakyaPasdJrVi7xvr5sxeM
```

This will return the print statements to stdout made by the mnist.py script. The entire process should take under 5 minutes.

You will see the following output while the model is training on the train data

```
Epoch 2/10
 32/60000 [..............................] - ETA: 13s - loss: 0.0140 - acc: 1.
352/60000 [..............................] - ETA: 9s - loss: 0.0704 - acc: 0.9
704/60000 [..............................] - ETA: 9s - loss: 0.0630 - acc: 0.9
1056/60000 [..............................] - ETA: 9s - loss: 0.0666 - acc: 0.9
```

When the training process is completed, you will see the accuracy score printed as below:

```
score = 0.9923
```

References:

[1] https://elitedatascience.com/keras-tutorial-deep-learning-in-python
[2] www.deepbrainchain.org