The following summary consist of 3 parts:

1. Network structure

Part 1 gives a model.summary() print from mDNN.

1. Model parameters in mDNN

Part 2 illustrates the different parameters and layers used in mDNN. A table for the parameters used for testing the mDNN ResNet would be given at the end of this section.

1. Result of training on CIFAR10

Part 3 summarizes the result of using different settings for learning rate, initialization and batch normalization to train the ResNet on CIFAR10 dataset classification problem.

1. Network structure

The network structure uses a ResNet structure as follows:

Name Output Shape # Params

========================================================

InputLayer (32, 3, 32, 32) 0

Conv2d (32, 64, 32, 32) 1728

BatchNormLayer (32, 64, 32, 32) 0

RELU (32, 64, 32, 32) 0

Conv2d (32, 64, 32, 32) 36864

BatchNormLayer (32, 64, 32, 32) 0

RELU (32, 64, 32, 32) 0

Conv2d (32, 64, 32, 32) 36864

BatchNormLayer (32, 64, 32, 32) 0

RELU (32, 64, 32, 32) 0

Conv2d (32, 64, 32, 32) 36864

BatchNormLayer (32, 64, 32, 32) 0

RELU (32, 64, 32, 32) 0

Conv2d (32, 64, 32, 32) 36864

BatchNormLayer (32, 64, 32, 32) 0

RELU (32, 64, 32, 32) 0

Conv2d (32, 128, 16, 16) 73728

BatchNormLayer (32, 128, 16, 16) 0

RELU (32, 128, 16, 16) 0

Conv2d (32, 128, 16, 16) 147456

BatchNormLayer (32, 128, 16, 16) 0

Conv2d (32, 128, 16, 16) 8192

BatchNormLayer (32, 128, 16, 16) 0

RELU (32, 128, 16, 16) 0

Conv2d (32, 128, 16, 16) 147456

BatchNormLayer (32, 128, 16, 16) 0

RELU (32, 128, 16, 16) 0

Conv2d (32, 128, 16, 16) 147456

BatchNormLayer (32, 128, 16, 16) 0

RELU (32, 128, 16, 16) 0

Conv2d (32, 256, 8, 8) 294912

BatchNormLayer (32, 256, 8, 8) 0

RELU (32, 256, 8, 8) 0

Conv2d (32, 256, 8, 8) 589824

BatchNormLayer (32, 256, 8, 8) 0

Conv2d (32, 256, 8, 8) 32768

BatchNormLayer (32, 256, 8, 8) 0

RELU (32, 256, 8, 8) 0

Conv2d (32, 256, 8, 8) 589824

BatchNormLayer (32, 256, 8, 8) 0

RELU (32, 256, 8, 8) 0

Conv2d (32, 256, 8, 8) 589824

BatchNormLayer (32, 256, 8, 8) 0

RELU (32, 256, 8, 8) 0

Conv2d (32, 512, 4, 4) 1179648

BatchNormLayer (32, 512, 4, 4) 0

RELU (32, 512, 4, 4) 0

Conv2d (32, 512, 4, 4) 2359296

BatchNormLayer (32, 512, 4, 4) 0

Conv2d (32, 512, 4, 4) 131072

BatchNormLayer (32, 512, 4, 4) 0

RELU (32, 512, 4, 4) 0

Conv2d (32, 512, 4, 4) 2359296

BatchNormLayer (32, 512, 4, 4) 0

RELU (32, 512, 4, 4) 0

Conv2d (32, 512, 4, 4) 2359296

BatchNormLayer (32, 512, 4, 4) 0

RELU (32, 512, 4, 4) 0

FlattenLayer (32, 8192) 0

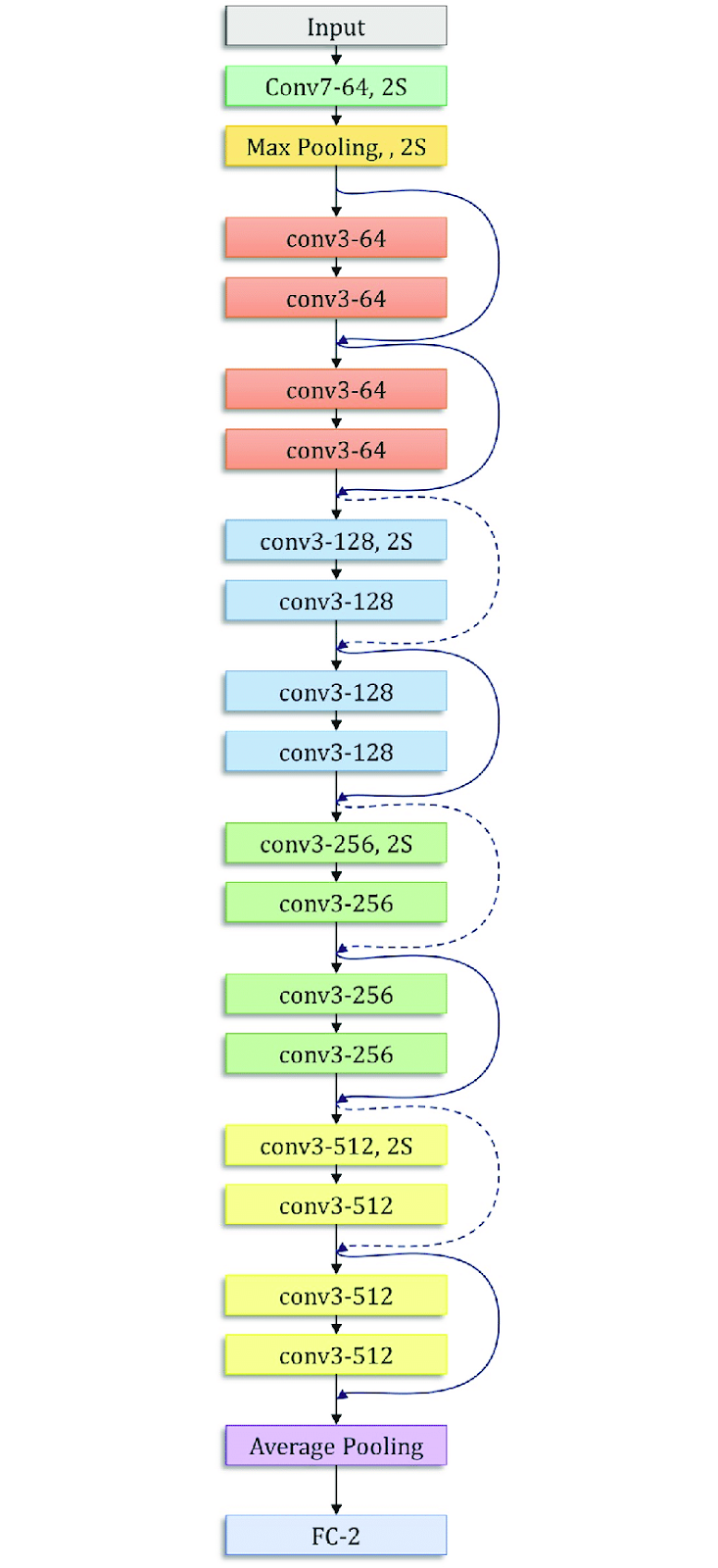
FullyConnected (32, 10) 81930

SOFTMAX (32, 10) 0

OutputLayer (32, 10) 0

========================================================

Total number of params : 11241162



1. Model parameters in Magmadnn

Different parameters have to be set in the MagmaDNN model before training. In this section, settings used when training the ResNet on CIFAR10 dataset will be discussed.

They are:

* Weights initialization
* Optimizer
* Loss function
* Learning rate
* Batch size
* Number of epoches

There are also layer-specific features to be discussed. The following layers used in building ResNet will be discussed:

* Convolutional Layers
* Batch Normalization Layers
* Fully Connected Layers
* Flatten Layers

At the end of this section, a summary on parameters that we used to train the mDNN resnet on CIFAR10 dataset will be given.

***Parameters for training***

Weight initialization

For weight initialization in 2D Convolution layers, we used the famous Xavier (Glorot Uniform) Initialization.

First, we define:

fan\_in : n\_feature\_maps\_in (channel in) \* rececptive\_field\_size (kernel\_size)

fan\_out = n\_feature\_maps\_out (channel out) \* rececptive\_field\_size (kernel\_size)

The idea of Glorot Uniform Initialization is to draw weight from a uniform distribution with range depending on fan\_in and fan\_out in order to stabilize the variance of the activation variances and back-propagation variance as one moves ups and down of the network. The weights are draw from the uniform distribution:

where and are the fan\_in and fan\_out of the current layer.

Readers may refer to <http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf> for more details.

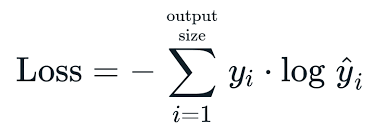
Optimizer

For the optimizer, we used the SGD, which calls the gradientdescent.h file. It is using the math/optimizer\_math/sgd\_momentum.h to do the update of the parameters. The gradient descent is with momentum, and the formula for updating the weights are:

The default momentum is 0.9 and the learning\_rate has to be defined by the user. The effect of learning of the performance of mDNN will be discussed together with the result in later sections.

Loss Function

For the loss function we used the Cross\_entropy where



The cross entropy calculations in MDNN are done through CuDNN calls.

Learning rate

For the learning rate, a different learning rate has been used to train the ResNet with CIFAR10. In mDNN, the learning rate is constant throughout training. Result will be presented in a later section. Generally, set .

Batch size

Because of limitations of memory, we have only used a smaller batch size of 32 to train mDNN ResNet, whereas in the original paper of ResNet18, they used a larger batch size of 256.

Number of Epochs

Generally, we have trained the ResNet with about 50 epochs to allow convergence.

**Layers Specifications**

In the ResNet of mDNN, excluding activation functions layers such as ReLU and softmax, we have used mainly 5 types of layers. They are : 2D Convolutional Layers, Batch Normalization Layers, Fully Connected Layers, Flatten Layers and Pooling Layers.

The fully connected layer, pooling, and flatten layers would not be discussed since there is little ambiguity between mDNN, TensorFlow and Pytorch.

2D Convolutional Layer

In mDNN, the class definition of 2D Convolution layer is as follows:

Conv2d (op::Operation<T>\* input, const std::vector<unsigned int>& filter\_shape, int out\_channels, const std::vector<unsigned int>& padding, const std::vector<unsigned int>& strides, const std::vector<unsigned int>& dilation\_rates, bool use\_cross\_correlation, bool use\_bias, tensor\_filler\_t<T> filter\_initializer, tensor\_filler\_t<T> bias\_initializer)

That is, after removing the data types :

Conv2d(input,filter\_shape, out\_channels, padding, strides, dilation\_rates, use\_cross\_correlation, use\_bias,filter\_initializer, bias\_initializer)

Remarks:

1. Currently mDNN conv2d layer does not support bias.
2. The default is use\_cross\_correlation = True. For details refer to [Convolution and cross-correlation in neural networks - PyImageSearch](https://www.pyimagesearch.com/2021/05/14/convolution-and-cross-correlation-in-neural-networks/)

Batch Normalization Layer

In mDNN, the batch normalization is done by calling the batchnorm\_device in cudnn.

There are some parameters for the batch normalization layer. They are :

* moving\_mean\_initializer : default = 0
* moving\_varience\_initializer : default = 0
* saved\_moving\_mean : movining\_mean and moving variance are updated during training, and saved to saved\_moving\_mean for the model to use during inference.
* saved\_varience : similar to saved\_moving\_mean
* bn\_scale : default = 1
* bn\_scale\_diff : default = 1
* bn\_bias: default = 0
* bn\_bias\_diff : default = 1

The batchnorm layer follows each convolution layer and helps stabilize the network during training.

**Parameters summary of ResNet of mDNN training with CIFAR10**

| Parameters |  |
| --- | --- |
| Weights initialization | Xavier (default in mDNN) |
| Optimizer | SGD |
| Loss function | Cross entropy |
| learning rate | depends, usually within 0 to 1 |
| Batch size | 32 |
| Number of epoches | depends, usually 50 |

1. Result on CIFAR10

Here we conclude three different cases:

* with ONES initialization and Xavier initialization of weights
* with old batchnorm (bn\_bias = 1) and new batchnorm(bn\_bis = 0)
* with higher learning rate and lower learning rate

Lastly, we directly compare the performance of our model with ResNet in Pytorch

with ONES initialization and Xavier initialization of weights

( With learning rate = 0.001, old batch normalization layer )

| Model | ONES initialization | Xavier initialization |
| --- | --- | --- |
| Accuracy after 5 epochs | 0.3288 | 0.4249 |
| Accuracy after 50 epochs | 0.5292 | 0.8603 |

with old batchnorm (bn\_bias = 1) and new batchnorm(bn\_bis = 0)

( With learning rate = 0.001, Xavier initialization )

| Model | old batchnorm | new batchnorm |
| --- | --- | --- |
| Accuracy after 5 epochs | 0.4249 | 0.6252 |
| Accuracy after 50 epochs | 0.8574 | 0.9601 |

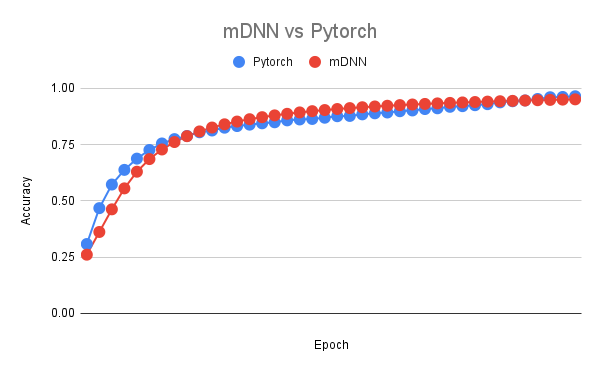
with higher learning rate and lower learning rate

(With Xavier initialization and new batch normalization layer)

| Model | lr = 0.001 | lr = 0.01 |
| --- | --- | --- |
| Accuracy after 1 epochs | 0.2603 | 0.3029 |
| Accuracy after 5 epochs | 0.6252 | 0.5744 |
| Accuracy after 40 epochs | 0.9505 | 0.9334 |

with ResNet in Pytorch

| Model | Pytorch | mDNN |
| --- | --- | --- |
| Accuracy after 5 epochs | 0.6876 | 0.6295 |
| Accuracy after 40 epochs | 0.9645 | 0.9509 |



Every run will be different as the initialization is random, but will tend to have results in the same region.

You can find the training data [here](https://docs.google.com/spreadsheets/d/1u2NbRxtTHRVeNOHOExnOL1IpKGWCfWEItKrTs6pULGQ/edit?usp=sharing)