Report: Deep Complex Networks validation

Table of Contents

- Overview:
    - Motivation/Observation:
    - Hypothesis/Predictions:
    - Experimental design:
    - Findings:
- Technical details:
    - Theoretical claims and algorithms:
    - Tasks and datasets:
    - Experimental protocol:
    - Software & Hardware details:
- Results:
        - CIFAR-10 experiments:
        - CIFAR-100 experiments:
        - Discussion - conclusions and potential extensions:
    - References:

Overview:

Motivation/Observation:

After several iteration, the UpStride engine has improved considerably in speed and available features. Unit tests are in place to make sure each module behaves as expected, but it is critical to also perform high-level tests to make sure the full training and evaluation pipeline is correct. *Deep Complex Networks* (DCN) by Trebelsi et al. [1] is one of the seminal papers in hyper-complex deep neural networks and has a open source implementation [2]. The goal of this project is to validate our *type1* (complex) engine by reproducing the image classification results of [1] on CIFAR-10 and CIFAR-100.

Hypothesis/Predictions:

Hypothesis 1: An implementation of DCN using the *UpStride Engine* will achieve a statistically comparable accuracy with respect to original public implementation [2] on the CIFAR-10 experiments described in the paper [1].

Hypothesis 2: An implementation of DCN using the *UpStride Engine* will achieve a statistically comparable accuracy with respect to original public implementation [2] on the CIFAR-100 experiments described in the paper [1].

Experimental design:

We focus on reproducing the image classification experiments using the *WideShallow* architecture described in [1] on two classical computer vision benchmark datasets:

- CIFAR-10 [3]: 10 classes, 50,000 training images, and 10,000 validation images
- CIFAR-100 [3]: 100 classes, 50,000 training images, and 10,000 validation images

We estimate that reproducing the results with other architectures/datasets for image classification brings limited additional information, considering that they all test the same modules of our engine. Also, the NLP-related experiments described in [1] make use of recurrent layers, which are not implemented in our engine.

Findings:

Our experiments confirm both hypotheses showing that we can reproduce the results of the original DCN implementation with our engine. These results validate the parts of our engine related to the implementation of complex-valued convolutional networks. We can therefore safely build on top of it for future research and commercial projects.

Technical details:

Theoretical claims and algorithms:

Since this is simply a validation study, there are no significant theoretical and algorithmic novelties that are worth discussing. We rather focus on comparing the manuscript and the code of DCN to expose potential differences and highlight details that are omitted from the paper (see Table 1).

	DCN paper	**DCN code**
Optimizer	SGD w/ Nesterov, momentum = 0.9	They allow SGD, Nesterov, Adam and RMSPros
Training epochs	fixed at 200	same as the paper
Learning rate schedule	Step-wise scheduler: • lr 0.01 at epochs 0-9 • lr 0.1 at epochs 10-99 • lr 0.01 at epochs 100-119 • lr 0.001 at epochs 120-149 • lr 0.0001 at epochs 150-199	same as the paper
Architectural topology	Each architecture is composed of **3 stages**, each containing several residual blocks which varies depending on the chosen architecture (WS, DN, IB). The residual blocks use **3x3 conv kernels** throughout the network. Between stage 1&2 and stage 2&3 the number of filters doubles and the resolution reduces by a factor 2. the doubling of the filter is done by concatenating the output of the last residual block with the output of a 1x1 convolution applied on it with the same number of filters used throughout the stage. The downsampling strategy is *not specified*. Before feeding the input to the first residual block, the following block is used: *Conv BN ReLU* Note that complex networks also have the additional *l earnVectorBlock* before this first processing block.	Essentially the same described in the paper. The downsampling strategy is conv w/ stride=2 (see ‘projection’ in the row ‘residual blocks’ for details).
WideShallow backbone	Real: • 18 real filters per conv in the 1st stage • 14 residual blocks per stage Complex: • 12 complex filters per conv in the 1st stage • 16 residual blocks per stage	Customizable with arguments
Final classifier	GAP > fully connected > softmax	Essentially the same as the paper. They use AveragePooling2D(pool_size=(8, 8)) to perform the GAP because when the code was written GAP was not available in Keras.
Regularization	• gradient clipping at 1.0 • no other regularization paper (not even weight decay/l2) is mentioned in the paper!	• gradient clipping at 1.0 • l2 regularization on all the conv and dense layers w/ weight 10^-4
learnVectorBlock (block to map the real part to extended real/complex /quaternion)	BNReLUConvBNReLUConv No other details are specified	BNReLUConv(3,1x1)BNReLUConv(3,1x1) • no bias • he_normal initialization • l2 kernel regularization with weight 10^-4

Residual blocks	x_n+1 = (BNReLUConvBNReLUConv)(x_n) + x_n	• regular: x_n+1 = (BNReLUConv(3x3)BNReLUConv (3x3))(x_n) + x_n • projection: x_n+1 = concat(BNReLUConv(3x3, stride=2) BNReLUConv(3x3))(x_n) , Conv(1x1, stride=2)(x_n))

Table 1: Comparison between the manuscript [1] and the code [2] of "Deep Complex Networks", with focus on optimization parameters and the WideShallow network architecture for image classification.

Our inspection shows that there are no substantial differences between the code and the manuscript. However, the code reveals two critical design choices for the network architecture, as well as the use of L2 regularization that were not mentioned in the paper.

Tasks and datasets:

CIFAR-10 is one of the most used datasets in computer vision and is the to-go option for benchmarking image classification methods. It is composed of 60,000 32x32 color images, split in a training set of 50,000 and a validation set of 10,000 images. Each image is assigned 1-out-of-10 possible: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. The classes are perfectly balanced, with 5,000 training samples and 1,000 validation samples per class.

CIFAR-100 is another popular dataset in computer vision for benchmarking image classification methods. This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. For example, the superclass flowers includes the classes orchids, poppies, roses, sunflowers, tulips. For the sake of our experimental evaluation, we follow the most common practice and consider the 100 classes independently of each other, i.e. without any consideration for the superclasses.

Experimental protocol:

With our experimental evaluation, we aim at validating our engine by reproducing a set of experiments from [1] and checking if we can obtain the same results produced by their publicly available code [2]. Please note that the goal is not to validate the full set of experiments of the paper, but rather validate our own code by comparing it to an external, well-respected implementation. For this reason, we only focus on reproducing the results of their complex-valued WideShallow network (with complex batch normalization and complex independent initialization) on the image classification task with CIFAR-10 and CIFAR-100.

Why not using other architectures proposed in the paper?
The *WideShallow* architecture is reported to be the best tested architecture. The other alternatives, *DeepNarrow* and *InBetween*, have different number of residual blocks and filters, but use exactly the same operations, therefore they bring no additional value from a code validation standpoint.

Why not testing tasks other than image classification?
The other two tasks discussed in the paper, automatic music transcription and speech spectrum prediction, are tackled with recurrent neural networks. Since we do not have an implementation for recurrent layers in our engine and do not priorities these tasks at the moment, they are out of the scope of this validation.

To obtain the results reported in the following section, we run the public implementation of DCN with the hyper-parameters reported in the paper and indicated in Table 1. We cannot consider the hyper-parameters found by Trebelsi et al. to be the best for us since their code is based on Theano and a former version of Keras, while ours is based on TensorFlow 2 (see study on difference between deep learning framworks [4]). For our implementation, we run a small grid search on the validation set of CIFAR-10 for batch size (BS= {32, 64, 128}) and weight decay (WD= {1e-4, 5e-4}). All the hyper-parameters and the data processing pipeline are the same in the two implementations, excluding those indicated in Table 2.

Key hyper-parameters used to obtain the final reported results		
Parameter	**DCN code**	**ours**
Batch Size (BS)	64	128
Weight Decay (WD)	1e-4	5e-4

Table 2: Difference in hyper-parameters between the experiments with our engine (ours) and the original code for "Deep Complex Netwoks" (DCN code) [2] .

Please note that this small grid search has not been repeated for CIFAR-100, for which we simply reuse the set of parameters found on CIFAR-10.

For each of the considered experimental settings, we report the mean and standard deviation of the validation accuracy over three runs, as well as the number of distinct trainable parameters, the runtime allocated memory, and the FLOPS count. Please note that we consider the accuracy of the last epoch, not the maximum accuracy achieved during the training.

Software & Hardware details:

Software:

- Repository: https://github.com/UpStride/classification-api/tree/experiment/papers
- Commit: `dd2055eef60499d49cd82c187ea9c2f63ac7277c`
- Docker image: tensorflow/tensorflow:2.4.0-gpu
- Upstride Python Engine Repository: https://bitbucket.org/upstride/upstride_python/src/master/
- Commit: `4669f1aa35519b3614d73f47e799011a2fce4c62`

Hardware 1:

- Type: UpStride Server Gaia / Hyperion
- GPU: TITAN RTX (x1) / Quadro RTX 6000 (x1)
- CPU: up to 32 cores / up to 80 cores

Hardware 2:

- Type: Oracle/Azure/Scaleway Virtual Machine
- GPU: Tesla V100 (x1) / Tesla P100 (x1)
- CPU: up to 6 cores

Results:

Table 3 and Figure 1 summarizes the results of our experiments. Please proceed to the next paragraphs for detailed discussions on these results.

	Evaluation on	**CIFAR-10**	**CIFAR-100**
DCN paper [1]	test set	93.83	73.64
DCN code [2]	validation set	94.27 ± 0.18	71.26 ± 0.48
	test set	93.38 ± 0.19	71.46 ± 0.18
ours	validation set	94.18 ± 0.14	71.59 ± 0.56
	test set	93.42 ± 0.04	71.28 ± 0.40

Table 3: Accuracy (%) of our engine (ours) and the original code for “Deep Complex Netwoks” (DCN code) [2]. As a reference, we also indicate the results reported in the “Deep Complex Netwoks” paper [1] (DCN paper).

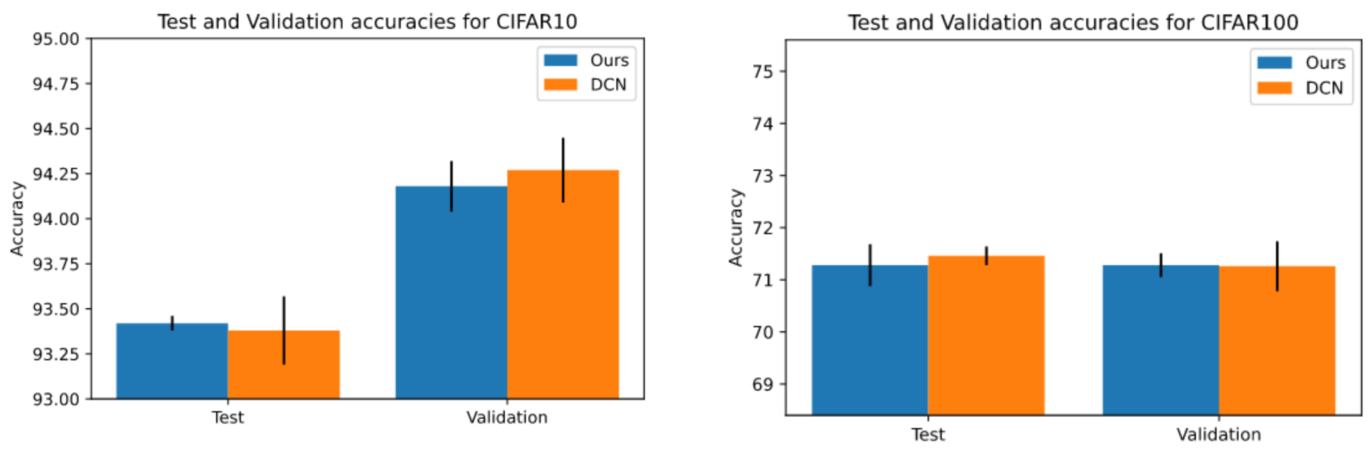

Figure 1: Barplot with error bars of the validation and test accuracy (%) of our engine (ours / blue) and the original code for “Deep Complex Netwoks” (DCN / orange) [2]. Note that the difference in accuracy between the two implementations is within the error bars.

CIFAR-10 experiments:

The results in Table 2 show that our implementation achieves statistically comparable accuracy to the original code, both on the validation and on the test set. However, the results reported in the paper are ~0.4% higher than both implementations. This difference might be cause by Trebelsi et al. using a different experimental protocol than us: this information is not provided in the paper.

CIFAR-100 experiments:

The results in Table 2 show that our implementation achieves statistically comparable accuracy to the original code, both on the validation and on the test set. However, the results reported in the paper are significantly higher (~2.2%) than both implementations. We do not have a concrete idea of what can have caused this difference since we are running supposedly the same code they used to obtain their results. Since we are focused on comparing our engine with their code, further investigating the difference with the paper is out of the scope of this project.

Discussion - conclusions and potential extensions:

In these experiments, we validated an implementation of the “Deep Complex Networks” [1] using our engine against the publicly available code [2] provided by the authors. The goal was to sanity check the modules in our engine necessary to implement a complex-valued deep convolutional neural network for image classification. More specifically, we wanted to verify the following hypotheses:

- *Hypothesis 1:* An implementation of DCN using the UpStride Engine will achieve a statistically comparable accuracy with respect to original public implementation [2] on the CIFAR-10 experiments described in the paper [1].
- *Hypothesis 2:* An implementation of DCN using the UpStride Engine will achieve a statistically comparable accuracy with respect to original public implementation [2] on the CIFAR-100 experiments described in the paper [1].

To verify *Hypothesis 1,* we compare the validation and test accuracy of our and the original implementation CIFAR-10. Both validation and test accuracy are statistically equivalent for the two implementations. We therefore conclude that ***Hypothesis 1* is verified**.

To verify *Hypothesis 2,* we compare the validation and test accuracy of our and the original implementation CIFAR-100. Both validation and test accuracy are statistically equivalent for the two implementations. We therefore conclude that ***Hypothesis 2* is verified**.

A closer look at the results in Table 3 reveals that the results of both implementations are lower than the results reported in the. This difference is particularly significant for CIFAR-100. We believe that this difference might be caused by unreported practices in the experimental protocol. Further inspection of this issue is out of the scope of this project.

In conclusion, our experiments validate the parts of our engine related to the implementation of complex-valued convolutional networks. Given the generic implementational strategy adopted in our code, these results give us more confidence on the engine as a whole.

References:

1. Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, Christopher J Pal. “Deep Complex Networks”. In *Internation Conference on Learning Representations (ICLR),* 2018
2. https://github.com/ChihebTrabelsi/deep_complex_networks
3. https://www.cs.toronto.edu/~kriz/cifar.html
4. Liu, Ling, Yanzhao Wu, Wenqi Wei, Wenqi Cao, Semih Sahin, and Qi Zhang. "Benchmarking deep learning frameworks: Design considerations, metrics and beyond." In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1258-1269. IEEE, 2018.