Testing Mixed-Precision for VGG, Inception and ResNet on the Dogs vs. Cats Dataset

Ruben Steendam
Artificial Intelligence, Utrecht University

Supervisor: Dr. M. van Ommen
Department of Information and Computing Sciences, Utrecht University

Second Supervisor: Dr. J.D. Dotlačil
Utrecht Institute of Linguistics OTS, Utrecht University

Bachelor Artificial Intelligence, Utrecht University, 7.5 ECTS

June 19, 2020

Abstract

Mixed-precision floating points seem promising in reducing computation costs for deep neural networks. But does the technique live up to the promise and do all network architectures benefit equally from mixed-precision? Using the Dogs vs. Cats dataset we researched the effect of using mixed-precision on VGG, Inception and ResNet by measuring accuracy, training speed and inference speed. The results showed that the accuracy of mixed-precision was comparable with that of single-precision. Furthermore, all networks became faster, both in training and inference. The speedup between the different architectures varied between 32% and 45% for training and between 10% and 40% for inference.

1 Introduction

Deep neural networks are well known for their great performance in areas such as image recognition with famous network architectures like VGG, Inception and ResNet. A downside of the great performance is the high computation costs of these deep networks. The holy grail for neural networks is to reduce this computation costs without reducing the performance.

NVIDIA proposed the so called mixed-precision format to reduce these computation costs. This technique changes the precision of the calculations from single-precision floating-point to mixed-precision floating-point. Mixed-precision floating-point uses a mix of single and half-precision floating-points. Using half-precision leads to lower computation costs. However the less precise half-precision floating-points also reduce the accuracy of the model. NVIDIA claims that the mix of half- and single-precision floating points ensures that the accuracy of the network does not decrease, while the computation costs are reduced.

This mixed-precision format is introduced in

2017 by NVIDIA. Very little research towards this format has been conducted. NVIDIA claims that this technique will improve the speed of neural networks, but to maximize the speedup a new generation of NVIDIA GPUs is introduced. NVIDIA might therefore claim too much and their paper written by Micikevicius et al. (2017) could be biased. We will test this floating-point format to answer the following research question: *Do VGG, Inception and ResNet all benefit equally from mixed-precision?*

These networks were chosen because of their variance in architecture and size, while all are well-known classification networks. Furthermore, they all are available in Keras, pre-trained on ImageNet. Thus by testing these three networks the NVIDIA claim for mixed-precision is put to the test.

To test the claims of accuracy and speed, we will look into the Dogs vs. Cats dataset from Kaggle. We will train and compare the three networks in both single- and mixed-precision floating-point. These six resulting models will be compared on accuracy, training speed and inference speed.

This paper gives in Section 2 some necessary background information for a better understanding of the problem and experiments. Section 3 explains the experimental setup for the comparison of the different floating-point formats. The results are split into three aspects: accuracy, training speed and inference speed and are presented in Section 4, after which they are evaluated and discussed in Section 5. The relevant conclusions will be drawn and given in Section 6.

2 Related work

2.1 Neural network calculations

There have been a number of publications about optimizing neural networks. Before we can go over these papers it is necessary to first understand how neural networks work and what operations they use.

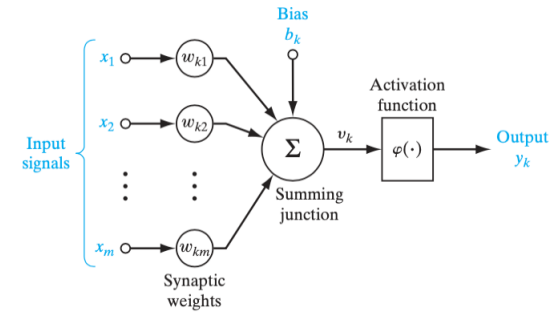
Neural networks are made up of multiple layers of neurons, see Figure 1 for a schematic drawing. The mathematical definition of a neuron, given by Haykin et al. (2009), is as follows:

$$f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

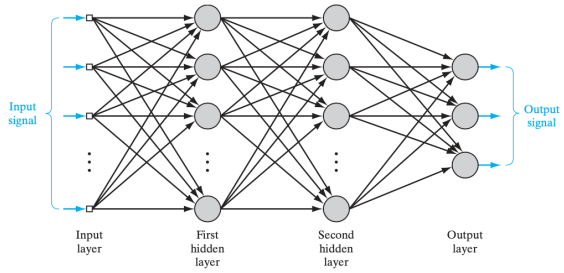
Here f is the activation function, b is the bias, x_i is the input on the ith element and w_i the weight for that element. All variables are scalars. Thus the sum is simply a vector-vector multiplication. Figure 1b shows that a neural network is nothing more than multiple layers consisting of multiple neurons. Since a layer consists of multiple neurons, all of these vector-vector multiplications together can be seen as matrix multiplications. The high amount of neurons and layers leads to a neural network executing many matrix multiplications. These matrix multiplications on its own are not that hard to perform. But the sheer size of deep neural networks requires many millions of multiplications, resulting in high computation costs. The relative simplicity of the individual calculations and the high number of calculations are the reason why the parallel capabilities of a GPU work really well for neural network calculations. To reduce the computation costs for neural networks, one can reduce the number of calculations, known as pruning, or reduce the complexity of each calculation, known as quantization.

2.2 Single-precision floating-point

Computers use different formats for number representation. As defined by Micikevicius et al. (2017) the default representation for neural network calculations is single-precision floating-point, also known as float32. This is a floating point number consisting of 32 bits.



(a) Nonlinear model of a neuron, labeled k



(b) Architectural graph of a multilayer perceptron with two hidden layers.

Figure 1: Figure 1a shows the internals of a neuron. Figure 1b shows a multi layer perceptron. Figures from Haykin et al. (2009)

Floating point numbers consist of the sign (s), the fraction (f) and the exponent (e) as described by Li and Chu (1997). See below for the formula.

$$(-1)^s \times \left(2^{e-127}\right) \times (1.f)$$

The sign is to indicate whether the number is positive or negative, the fraction contains the number's digits and the exponent tells us where the decimal point is placed. A float32 has 1 bit for the sign, 8 bits for the exponent and 23 bits for the fraction. A float16 has 1 bit for the sign as well but 5 bits for the exponent and 10 bits for the fraction. Reducing the number of bits reduces the computation costs of matrix multiplications and thus speeds up the neural network. The precision of the calculations decreases however. This is a drawback often found when optimizing performance of neural networks. One has to balance accuracy and speed.

2.3 Optimization techniques

As said before, broadly spoken, there are two ways to optimize existing neural networks. An alternative to these optimization techniques is to build efficient networks, either by building smaller networks or by using optimized structures. However this cannot be applied to existing networks architectures. Crowley et al. (2018) found that smaller networks with more training consistently outperformed pruned networks when looking at residual networks. Since this technique only works for new networks, where you design and build the network from the ground up, it discards transfer learning as a whole. Transfer learning is the technique where an existing trained network is used as a feature extractor. This base network is then adapted for your specific problem and retrained a little to process the new dataset.

Back to the two main techniques. Pruning is where you remove neurons from the network to decrease the number of calculations. And quantization is the technique of reducing the precision of the numbers and thus reducing the cost of one calculation. Something to watch out for is that neural networks are run in two phases, training and inference. Training is the first step for neural networks, where features are discovered and the network is learning with every example it processes. This learning is done by updating the weights in the model. Inference is when the network is trained and the model is deployed. This model is fed new data which the network only has to classify, without learning, so without weight updates. Optimization is almost always focused at inference, since training is done only once, where inference is the day-to-day appliance of your

program. However reducing training times can lead to a faster development of new networks, by allowing for quicker tests.

2.3.1 Pruning

Pruning can be split into two categories, structured and unstructured as stated by Anwar et al. (2015). Unstructured pruning has been studied by Turner et al. (2018), where individual weights were pruned, to reduce network size. Li et al. (2016) pruned whole convolutional filters, which reduced computation costs. The key to both forms of pruning is to select the weights or filters which contribute the least to the network's accuracy. There are several ways to find the least contributing weights. The two most well-known are that of l_1 pruning, tested by Li et al. (2016), and Fisher pruning from Theis et al. (2018). l_1 pruning is using the l_1-norm, which is the sum of the absolute weights of a filter. It selects the filters with the lowest l_1-norm and removes these. Fisher pruning is a more recent method and ranks filters according to the estimated change in loss that would occur on their removal. Crowley et al. (2018) demonstrated that Fisher pruning gives better results on Residual Networks, such as ResNet.

2.3.2 Quantization

Quantization studies the reduction of precision in neural networks. Single-precision floating-point (float32) is the default for neural networks, quantization reduces this to lower precision numbers such as half-precision (float16), int8 or even lower. The reduction in precision reduces the computation costs greatly as demonstrated by Zhou et al. (2017). An even more extreme form is binarization, where every weight and activation is reduced to being either 1 or -1 as shown by Courbariaux et al. (2014). Courbariaux et al. (2016) tried binarization on the MNIST dataset and reported an increase in performance without losing accuracy. However there is no evidence that this technique also works on bigger and more complex datasets, such as ImageNet.

Another advantage of quantization is that it becomes more feasible to run models on a CPU. GPUs are pretty good at single-precision calculations, but CPUs are not. As stated by Wu et al. (2016) quantization thus is useful for using models on embedded systems.

2.4 Mixed-precision

Micikevicius et al. (2017) introduced mixed-precision, a technique developed by NVIDIA. Mixed-precision is a quantization technique which reduces computation costs as normal quantization does, but without harming accuracy. This technique works by having a mix of single- and half-precision floating-points. In Figure 2 a schematic is given of a training iteration on a mixed-precision network. As can be seen the weights are saved as single-precision, and for every iteration the weights are converted to half-precision floating points. Then the whole network is ran with these half-precision weights and activations. After a pass, the weights gradient, still in half-precision, is used in the weight update function, which saves the weights as single-precision.

The weights are saved as single-precision floating points to distinguish between the very subtle differences in weights, which is something half-precision is not capable of. However, since almost all calculations and activations now take place in half-precision, the size and computations costs of the network are reduced, by nearly 50%, as activations account for nearly 90% of the total number of saved matrices, whereas weights account for only 10%. However not all types of layer can be reduced to lower forms of precision, batch normalization i.e. is never reduced to lower forms of

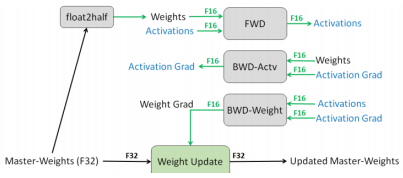


Figure 2: Mixed-precision training iteration for a layer. Figure from Micikevicius et al. (2017)

Model	Layers	Parameters
VGG	16	138M
Inception	159	24M
ResNet	152	60M

Table 1: The layers and parameters for VGG, Inception and ResNet. The M in parameters stands for million.

precision and will still use single-precision in mixed-precision.

Apart from the basic structure of the adaptations, NVIDIA applied loss scaling as well. Loss scaling is where the weight gradients are scaled so that they all fall in the half-precision range. Furthermore, mixed-precision requires specific new GPUs to benefit from the speedup. Given NVIDIA's commercial interest in selling these new GPUs, their view is likely to be biased, which is another reason for testing their technique.

2.5 Models

This paper compares single- and mixed-precision on the VGG, Inception and ResNet architectures. ResNet and Inception are well-known current state of the art networks with high accuracy scores and optimized network architectures in order to reduce computation costs. VGG is an older network which in its days was the best classification network available. Nowadays it is quite slow and it has lower accuracy scores than ResNet and Inception. All of these networks are convolutional neural networks, which are designed for image / video recognition.

Simonyan and Zisserman (2014) proposed VGG. VGG is a network consisting of 16 layers, which for the time was quite a lot. It has a total of 138M parameters and a top-1 accuracy of 0.713 and top-5 accuracy of 0.901 on the ImageNet dataset.

Szegedy et al. (2016) updated the original Inception network to V3. This network consists of 159 layers, with 24M parameters. Scoring a top-1 accuracy of 0.779 and a top-5 accuracy of 0.937. The Inception network is well known for its Inception modules. These Inception modules make a CNN naturally sparse. And sparse networks run faster. Inception modules work on the basis that it is hard to determine what size filter you need. So 1x1, 3x3 and 5x5 convolutions are done in parallel before moving to the next layer. To reduce the computation costs for the larger filter sizes a 1x1 convolution is done before the larger convolutions, which decreases the number of parameters. These Inception modules make it hard to make the network sparser.

He et al. (2016) proposed an update to the ResNet networks. The ResNet152V2 consists of 152 layers, with 60M parameters. Scoring a top-1 accuracy of 0.780 and a top-5 accuracy of 0.942. ResNets are designed around their residual connections. These connections combine input from the previous layer in a network with the n^{th} previous layer, thus essentially creating skip connections. This ensures that the gradient is maintained throughout the network, instead of getting an infinitely small gradient, which essentially becomes zero. In plain English, the skip connections combine higher and lower level features, to ensure that all features are detected and networks can grow even deeper.

3 Methods

The problem used for these experiments is that of the Dogs vs. Cats dataset from Kaggle (https://www.kaggle.com/c/dogs-vs-cats), where the goal is to recognize picture as either a dog or a cat. This dataset consists of a train and test set. The train set is labeled, but the test set is not. For this experiment the test set will be discarded and the train set will be split into a train and validation set, using an 80/20 split. For the neural network building and testing TensorFlow 2.2 will be used. This version of TensorFlow supports mixed-precision when used with the Keras API. Another benefit of using Keras is that it has pre-trained ImageNet models of well-known deep learning networks. The experiments will be run on a Google Cloud Platform Virtual Machine with an NVIDIA V100 to ensure that mixed-precision support is enabled. All programming will be done in Python 3.5.3.

To ensure that the experiments vary as little as possible, we will be initializing the pseudo-random generators with a seed of 42. Furthermore all networks will be loaded without the fully connected layer at the end of the model by using $include\_top = false$. This layer is designed for for ImageNet which has 1000 classes, where we will only need 2. The models need to be adapted to our problem. To do this three layers will be added. First a global average pooling 2d layer will be added, second a dense layer with 1024 nodes and lastly a fully connected dense layer with 2 nodes. This is one of the solutions for transfer learning as proposed by Lin et al. (2013), to ensure that the model will be capable of recognizing new classes. All layers except these last three will be frozen, since we want to use the features learned on ImageNet as a feature extractor. Only the later added classification layers will need to be trained for the Dogs vs. Cats problem.

All models will be trained two times. Once with single-precision and once with mixed-precision. After loading and adapting all models, every model is trained for up to eight epochs. Eight epochs was found to be the point where the models converged to their optimal training accuracy. The best model, based on training accuracy is selected and for this model the validation accuracy and loss will be reported. The training and inference speed will be reported as well. Unlike the accuracy tests where the base layers were frozen, the full model is run for the speed tests and all layers are trained. To ensure that the results are comparable all models get eight CPU workers and are given a batch size of 64, this eliminates the CPU as a bottleneck and makes the results comparable. Training/inference speed will be compared directly and reported in milliseconds. For the accuracy a McNemar's test will be conducted.

The McNemar's test is a statistical test, to check whether two models are significantly different. Every sample is evaluated by the single- and mixed-precision model. And for every sample it is noted whether the model correctly classified or not. Then a contingency table with correct/correct, correct/incorrect, incorrect/correct and incorrect/incorrect will be made. This data will be used for the McNemar's test, to see whether or not the null hypothesis, that the models are equal, can be rejected.

4 Results

The results for the single- and mixed-precision runs will be analysed on the aspects of accuracy, training speed and inference speed.

Table 2 shows the raw results for validation accuracy scores. Two things stand out. First, the difference in accuracy between single- and mixed-precision seems to be negligible and second both Inception and ResNet outperform

Model	Single-precision (%)	Mixed-precision (%)
VGG	91.70%	90.74%
Inception	98.46%	98.38%
ResNet	99.10%	99.12%

Table 2: The validation accuracy scores in % for VGG, Inception and ResNet both on single- and mixed-precision.

Model	Test Statistic	P-value
VGG	19	0.371
Inception	12	0.701
ResNet	10	0.832

Table 3: The results of the McNemar's test comparing the single- and mixed-precision variants of VGG, Inception and ResNet.

VGG . The results of the McNemar's test, as shown in Table 3 fails to reject the null hypothesis of single- and mixed-precision models being equal. Ergo, there is no significant difference in accuracy between single- and mixed-precision.

The single- and mixed-precision results for training speeds can be seen in Table 4. The difference in speed varies a bit between the models. Changing the precision for ResNet from single- to mixed-precision reduces the time per step by 45%. While Inception only reduces by 32%.

Finally in Table 5 the speed per inference step can be seen for all models on both single- and mixed-precision. Here the difference in speed is even more evident when compared to the training results. Where Inception only differs by 10%, ResNet differs by 40%, proving that the speedup of mixed-precision is model dependent.

Model	Single-precision (ms/step)	Mixed-precision (ms/step)	Speed difference (ms/step)	Speed difference (%)
VGG	303	196	107	35%
Inception	195	133	62	32%
ResNet	492	271	221	45%

Table 4: The training speeds for VGG, Inception and ResNet and the difference in training speed in %. A training step is the average of the time it takes to process one mini-batch. The difference is calculated by dividing the difference between single- and mixed-precision speed by the single-precision speed.

5 Discussion

The research question was *Do VGG, Inception and ResNet all benefit equally from mixed-precision?*. This was tested on the Dogs vs. Cats dataset. The results indicate that the accuracy does not decrease significantly, while the network performs faster in both training and inference. The speed difference however, does vary across the models quite a bit. This discussion will be split into three sections: Accuracy, Training speed and Inference speed.

5.1 Accuracy

As can be seen from Table 2, the validation accuracy differs by a maximum of 0.96 percentage point between single- and mixed-precision. Looking at the results from the McNemar's Test, see Table 3, the p-value shows that the differences in accuracy are indeed insignificant. And the accuracy is thus confirmed comparable.

Mixed-precision therefore seems like a good alternative for single-precision. It performs on an equal level as single-precision. This same level of accuracy was confirmed for all three models. This is likely due to the way mixed-

Model	Single-precision (ms/step)	Mixed-precision (ms/step)	Speed difference (ms/step)	Speed difference (%)
VGG	138	100	38	28%
Inception	97	87	10	10%
ResNet	180	108	72	40%

Table 5: The inference speeds for VGG, Inception and ResNet and the difference in inference speed in %. An inference step is the average of the time it takes to process one mini-batch. The difference is calculated by dividing the difference between single- and mixed-precision speed by the single-precision speed.

precision stores the weights. By allowing the model to save weights as single-precision floating points, subtle differences in weights could be saved. Further research could be conducted into half-precision to see whether half-precision does indeed reduce the accuracy significantly.

Of course the Dogs vs. Cats problem we used here is relatively simple. It is a binary image classification problem, where all three networks are pre-trained on ImageNet, which is a 1000 classes image classification problem. Further research could be conducted into more difficult image classification problems. Or one might even look into video recognition, speech recognition or even language problems such as machine translation. It could be interesting to see whether mixed-precision performs equally as good for these problems.

New models could be tested as well, this research only focused on classification networks. By using these three different classification models, the current assumption is that mixed-precision indeed works as designed on simple problems for convolutional classification models. All models, a simple and straightforward network like VGG and more complex networks such as Inception and ResNet showed the same behavior on mixed-precision with the same level of accuracy as single-precision.

5.2 Training speed

Looking at Table 4, one can see the training speed and difference in training speed between single- and mixed-precision for VGG, Inception and ResNet. ResNet is the slowest model to train, both on single- and mixed-precision, while VGG does consist of more parameters and no optimizations. This could be explained by the fact that ResNet has a lot more layers than VGG, while still having a sufficient number of parameters. This balance between layers and parameters could be the reason that ResNet is slower than Inception. The increase in speed is also the biggest for ResNet, both absolute and relative. Why ResNet receives a bigger speedup than VGG or Inception is unknown. It might be due to a latency limitation. This is that the CPU could not load the data fast enough to the GPU, so that the speedup by mixed-precision is limited. Since VGG and Inception are already faster than ResNet, the CPU has to work harder to load the data in time. Another reason could be that the impact of mixed-precision might be layer dependent or even depth dependent.

A depth dependent speedup however seems unlikely. Since the speed difference for VGG and Inception are in the same order of magnitude. Thus the cause of the difference might be due to the type of layers used. However all three networks use more or less the same layers: convolution, pool and fully connected. More research to the cause of this difference needs to be conducted.

5.3 Inference speed

Lastly we measured the inference speed for all three models on both single- and mixed-precision. Here it follows more or less the

same pattern as with training speed. ResNet is the slowest network, followed by VGG and then Inception. ResNet also gains most from changing to mixed-precision, followed by VGG and then Inception. And the differences in speedup are larger than on the training speed. ResNet's speed increases by 40%, where VGG differs by 28% and Inception only received a 10% speedup.

ResNet is also quite fast for inference compared to its training speed, with a general speedup of 63.41% for inference compared to training, compared to 54.46% for VGG and 50.26% for Inception. The cause of this difference in speedup could be found by a deeper network architecture analysis. This could be combined with the research to the cause of the training speed difference.

6 Conclusion

In short mixed-precision seems a promising and viable solution to manage the rising computation costs of deep neural networks. By testing three well-known classification models we showed that all three models performed equally with mixed-precision as with the default single-precision. The speed did increase for all three networks for both training and inference. Mixed-precision therefore delivers on its promise and can very well be a viable solution for deploying new networks or speeding up older architectures. However the speed differences between network architectures and between training and inference steps might need some further research to fully understand if there are limitations to mixed-precision and what those might be.

References

Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *arXiv preprint arXiv:1512.08571*, 2015.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Elliot J Crowley, Jack Turner, Amos Storkey, and Michael O'Boyle. Pruning neural networks: is it time to nip it in the bud? *arXiv preprint arXiv:1810.04622*, 2018.

Simon S Haykin et al. Neural networks and learning machines/simon haykin., 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Yamin Li and Wanming Chu. Implementation of single precision floating point square root on fpgas. In *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No. 97TB100186)*, pages 226–232. IEEE, 1997.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

Jack Turner, José Cano, Valentin Radu, Elliot J Crowley, Michael O'Boyle, and Amos Storkey. Characterising across-stack optimisations for deep convolutional neural networks. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 101–110. IEEE, 2018.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

A Code

For the code written for these experiments see https://github.com/RSteendam/quantization. After setting up your machine and dataset (using the bash scripts in script/). The program can be run by using run.py and selecting the type of experiment you want to run. See the readme.MD on Github for a further explanation.