Profiling Energy Consumption of Deep Neural Networks on NVIDIA Jetson Nano

Stephan Holly, Alexander Wendt, Martin Lechner
Christian Doppler Laboratory for Embedded Machine Learning
ICT, TU Vienna, Vienna, Austria
e1630033@student.tuwien.ac.at, alexander.wendt@tuwien.ac.at, martin.lechner@tuwien.ac.at

***Abstract*—Improving the capabilities of embedded devices and accelerators for Deep Neural Networks (DNN) leads to a shift from cloud to edge computing. Especially for battery-powered systems, intelligent energy management is critical. In this work, we provide a measurement base for power estimation on NVIDIA Jetson devices. We analyze the effects of different CPU and GPU settings on power consumption, latency, and energy for complete DNNs as well as for individual layers. Furthermore, we provide optimal settings for minimal power and energy consumption for an NVIDIA Jetson Nano.**

Keywords-power, energy, latency, NVIDIA, Jetson, deep neural network, profiling

I. INTRODUCTION

In many mobile applications, there is a need to achieve a certain accuracy within a defined latency. However, in mobile devices, power consumption must be considered as an additional constraint. Further, in some applications, latency is not the critical point, but power consumption or energy. A key to successfully optimize the power consumption of a neural network is to get accurate measurements of the actual power consumption on the device. For applications like object or vehicle detection from drone images [1] or in soccer-playing robot applications [2], low power consuming GPUs are a meaningful choice [3]. In this work, we will therefore look at the NVIDIA Jetson Nano.

The research question is to analyze how different hardware configurations, as well as network configurations, impact the power consumption as well as network latency of that board. Consequently, we can optimize the utilization of the NVIDIA Jetson Nano for given applications.

We use the following methodology: As a first step, we look for reliable ways to measure the power for the whole board as well as for single layers of a deep neural network by reviewing and testing some standard state-of-the-art methods. There are several methods available, e.g., measuring the power of existing sensors or using available internal tools on the board. The challenge is to interpret the results as background processes are considered.

Then, for a neural network, we vary hardware parameters like GPU and CPU frequency to see how to configure it for a particular application. Finally, we analyze how the configuration of neural networks affect power consumption. From that, we can conclude which hardware and network configurations have the highest impact on power consumption.

We contribute to research in this area in the following ways:

- We provide a measurement base for generating power estimation models like Neural Power [4]
- We provide insights into how specific hardware configuration settings, as well as network configurations, influence power consumption on the NVIDIA Jetson Nano
- We provide optimal configuration for the NVIDIA Jetson Nano of latency, energy- and power consumption per inference run

II. RELATED WORK

Different power measurement methods for embedded systems are discussed in [5]. Due to availability and accuracy, we are interested in measuring with shunt resistors.

SyNERGY [6] uses the onboard sensors of the Jetson TX1, and the ARM Streamline Performance Analyzer to measure the energy of inference at layer level. Furthermore, it creates a prediction model for energy. The precursor of SyNERGY is mentioned in [7], where it established fine-grained energy profiling on the Jetson TX1. In our case, we do not have access to ARM Streamline Performance Analyzer and will use only onboard sensors.

A power prediction framework is NeuralPower [4]. It uses the Nvidia System Management Interface `smi` for power measurements. This command-line utility reads out onboard sensors of the GPU, which measures the instantaneous power. Unfortunately, this tool is not available for the NVIDIA Jetson Nano platform.

In [8], the authors present an image recognition system that won the LPIRC prize in 2017. It describes the optimization steps that are required to achieve the best accuracy of the energy rating. The system is based on a Jetson TX1, and energy was measured with the WT310E power meter.

An overview of power loss measuring methods for power electronics is given in [9]. Calorimetric measuring methods, which are considered the most accurate measuring methods, were discussed. Due to the slow change of temperature and, therefore, long timing periods, this method does not seem to be feasible for our use case.

III. POWER MEASUREMENTS

For power measurements, we use the onboard sensors on the Nvidia Jetson Nano. Due to their limited resolutions and unknown characteristics, we verify the precision of the input sensor using reference measurements with an oscilloscope

and a shunt resistor. We introduce an experimental mapping method to link power measurements to the individual layers to get power and energy consumption for each layer. Then, we analyze the effect of power on different hardware settings. We focus on finding efficient power, latency, or energy settings for inference. Finally, we analyze the effect of network settings on power consumption.

A. Measurement Setup

The NVIDIA Jetson Nano is a low-cost development board to get started with neural networks. It shares a variety of functions with the other Jetson boards, which allows us to transfer software to another board quickly. NVIDIA TensorRT[1] is the main driver for running inference. It does not only run, but it also automatically optimizes the networks for the hardware. Furthermore, it can be used to create networks from scratch.

For power measurements, there are three onboard sensors installed, which are located at the power input of the board, the GPU, and the CPU. Unlike the NVIDIA Jetson TX2, there are no onboard shunt resistors for measuring power with an external energy probe. It limits the precision of the measurements to the onboard sensors.

The sensors can either be read automatically with the `tegrastats` tool or manually by reading sys files. `Sysfs` is a pseudo-file system on Linux. It exports various information to a virtual file system, which can be accessed. The sensor measurements can be found in the `I2C` folder. Power, voltage, and current can be read. The benefit of manually reading it is that one can achieve a higher time resolution, which is approximately 1.6ms. In terms of power resolution, 7mW is possible. Power logging was done by creating a process and then reading the sys-files for one of the sensors. After each reading, we measured the duration to be able to assign the value to an inference. In another process, we executed the inference and extended it with CUDA events. It allows us to measure the time for the start of loading the memory, inference start, and end, as well as the time after unloading the program from memory.

We use it to calculate the latency of inference, i.e., the difference between the start of loading memory to finished unloading. We measure latency with the power logger deactivated to eliminate added time from logging.

For most measurements, we use a pre-trained version of MobileNet V2 for ImageNet classification with an input resolution of 3x224x224. Additionally, for some measurements, we use tensors with random values. We import the network into TensorRT through the `OnnxParser` using the ONNX description format.

During measurements, we disconnect all peripheries except the ethernet port. It is used for communication over SSH. After the start of the measurement, we disable ethernet to decrease the effects on power further. We use Ubuntu 18.04 as the operating system from the Jetson Nano Development Kit SD Card image dated 17th of December 2019. We operate the OS in command line mode with the graphical user interface disabled, to reduce the number of background processes running during testing.

[1] https://developer.nvidia.com/tensorrt

B. Power Measurement Analysis

To verify the correctness of the sensors, we apply another method for power measuring. Due to missing onboard shunt resistors at the right nodes, only the input power sensor can be checked. For measuring, we put a shunt resistor into the power supply. We use an oscilloscope to probe each end of the resistor. The current through the resistor is calculated by dividing the voltage difference of the probes by the resistance value. The product of this current with the input voltage results in power consumption.

We perform two measurements: First, we measure the power with activated onboard sensors to see if the outputs are comparable. Then, we measure with sensors deactivated to see if sensing itself consumes power. In Fig. 1, we visualize those measurements. When the onboard sensors are deactivated, a difference of up to 1.0W can be detected, i.e., power is higher when measuring onboard. Measurements made on the Jetson Nano itself result in even lower power values. The electronics between input and sensor might consume this.

Besides measuring power for a whole inference cycle, a more in-depth view of each layer is interesting. For that, we use the TensoRT layer-time profiler, which returns the latency of each layer. We add these timings chronologically to the start time of each inference. With this information, we map every power value to a specific layer. A problem with this method is that the sum of layer-time from the profiler is unequal to the latency. It creates a time delay at the end where we cannot assign measurements to any layer. Depending on where the delay happens, wrong assignments could occur.

According to the TensorRT Python API, the profiler is called at the end of execution. Therefore, the delay should be in the end. Due to the low time resolution, this mapping can only be done with a certain number of inference cycles to get data for each layer. Depending on the time a layer takes, more values or fewer values can be assigned to a layer.

The number of inference cycles should, therefore, be set that the fastest layer is represented sufficiently. For MobileNetV2 the fastest layer is Shuffle. The number of inference cycles for mapping layers was set to 30000 to get at least 100 measurement points for the shuffle layer.

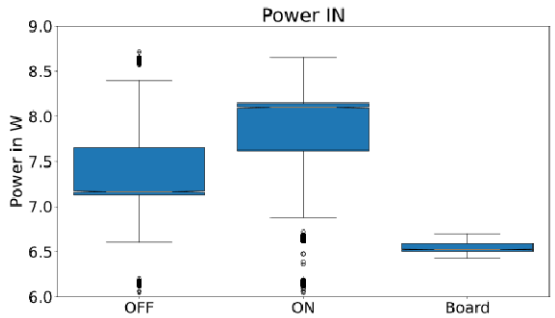

Figure 1. Power measurements of the internal and external power sensor

Fig. 2 shows the power consumption of each layer from MobileNetV2. Power stays nearly constant for all layers.

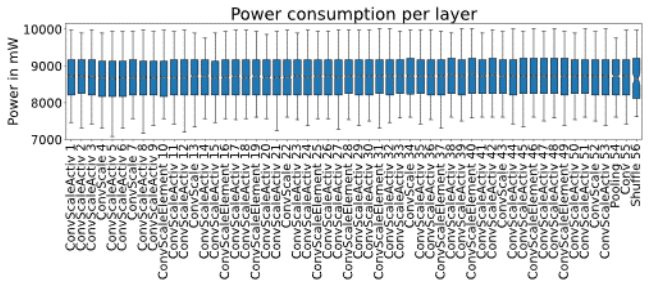

Figure 2. Power measurements over several cycles

We calculate the energy consumption for each layer by multiplying each mapped power value with the layer time. Fig. 3 shows the energy consumption for each layer. Due to the nearly constant power, the difference in energy is only caused by the difference in latency.

Although the sensors are limited in resolution and, as seen in Fig. 2, do need quite a bit of power. They provide sufficient values for further measurements. Furthermore, they supply information about CPU and GPU power that cannot be gathered otherwise.

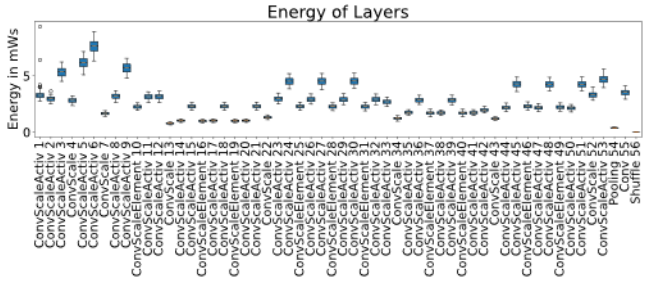

Figure 3. Energy measurements of single layers

C. *Impact of Hardware Settings*

For NVIDIA Jetson Nano, we can tune four hardware parameters: (1) Number of CPU cores used, (2) CPU frequency, (3) GPU frequency, and (4) Synchronization mode. Most of these parameters have been changed with the preinstalled `nvpmodel` [2] tool from NVIDIA. The basic settings, which have been set for all these measurements, if not otherwise mentioned, are the following: Network MobileNetV2 with input tensor of 3x224x224, batch size 1, all cores active and frequencies maximized to 1479MHz for the CPU and 921MHz for the GPU and synchronous mode.

The first parameter, which was analyzed, was the number of CPU cores. Jetson Nano has a total of four cores, which can be activated or deactivated. Fig. 4 shows the impact on power, energy, and latency. Power and energy rise with the number of cores, while latency stays the same.

[2] https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-321/#page/Tegra%2520Linux%2520Driver%2520Package%2520Development%2520Guide%2Fpower_management_nano.html%23wwpID0E0YI0HA

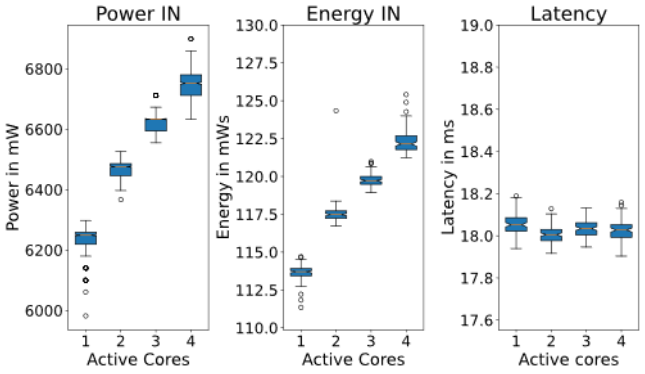

Figure 4. Power, energy, and latency as functions of the number of active cores

We achieve the lowest power- and energy consumption with only one active core. It increases the latency by 0.1% but provides power savings of approximately 700mW, i.e., a reduction of 7.4% power and 7% energy compared to the case with all cores active.

CPU frequency can be set from 102MHz to 1479MHz with an ~100MHz step size. In Fig. 5, we show the effect of this parameter. The stacked bar chart contains the power of CPU and GPU. Except for the first two frequencies, the power of the GPU stays nearly the same. However, below 307MHz, the dependency on CPU speed for inference gets noticeable. The combined power drops significantly from ~3.6W to ~2W. In the latency chart, this effect is also visible. Latency decreases with rising frequency. At the lowest frequencies, there is a significant increase in latency visible while the others have nearly the same value. Energy also shows a spike at the lowest frequencies. The remaining progression shows a linear curve, like the linear curve of CPU power.

When improving power, choosing the lowest frequency is the best setting, with an improvement of 45%. The tradeoff is an increase in latency, which doubles compared to the highest frequency. It makes total energy consumption increasing until it reaches its peak at 137mWs. For latency, the highest frequency is the best option. In case energy consumption is optimized, the frequency should set be to 307MHz. It has only a little impact on latency but reduces energy by 12%.

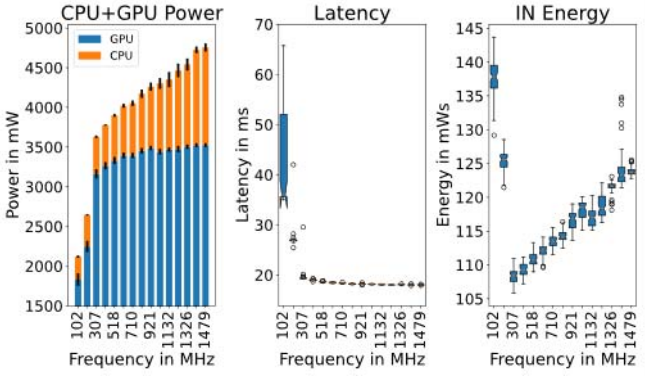

Figure 5. CPU+GPU power, latency, and energy frequency dependency

Another frequency, which can be adjusted is the GPU frequency. Its range is from 76MHz to 921MHz and can be set with a step size of ~77MHz. Fig. 6 displays this parameter. The most significant effect on power can be seen in the GPU sensor, where power rises from 400mW to 3500mW. There is a change in steepness visible at 614MHz, which leads to an increment of GPU energy after this point. CPU power also increases over the frequency. It is 27% higher at maximum frequency. Latency decreases substantially with the frequency. It drops from 181ms at the lowest frequency to 17ms at the highest. It means inference is executed ten times faster at 921MHz.

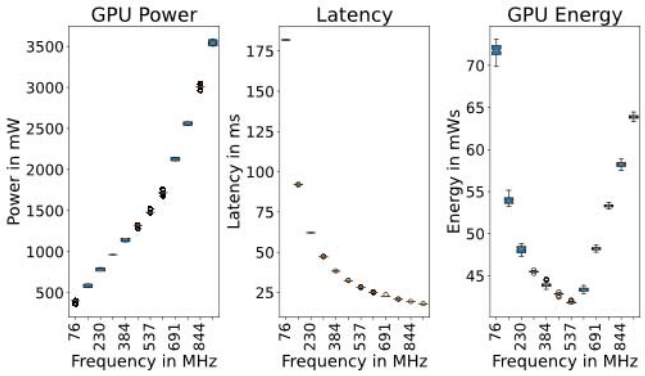

Figure 6. GPU power, latency and GPU energy dependency of GPU frequency

If power is optimized, then the lower GPU frequency, the better. Due to the minimum in GPU Energy at 537MHz, the total energy curve changes its track from falling to a nearly constant curve. The best energy value is achieved at 844MHz with 126mWs, but neighboring frequencies have nearly the same value.

If optimized for energy consumption, the frequency range from 537MHz up to 921MHz can be chosen without significant losses in latency. It allows us to choose a power-efficient mode with 537MHz or an optimized latency mode with 921MHz.

Asynchronous mode is a function from TensorRT, which allows us to perform inference in parallel. Memory copy, as well as CUDA kernel functions, get split up, and they are executed in a pipeline. It can result in faster execution of inference. The number of inputs to be executed in a sequence is defined by the batch size. The increased batch size can improve the results of the async mode. Synchronous mode, on the other hand, executes sequentially.

A feature that is only available for synchronous mode is the layer profiler, which gives information on how much time each layer needs. Fig. 7 shows the results of the async test. We executed each of the batch sizes, 1, 8, 16, 32, asynchronous as well as synchronous. We divided energy and latency by the batch size to get the relative value of a single input.

Overall, the power consumption for asynchronous mode is higher than for synchronous mode. For batch sizes greater than 1, an increment in the variance of the two modes can be spotted.

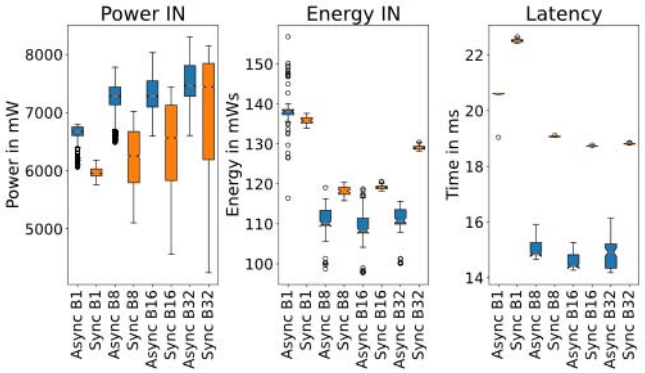

Figure 7. Asynchronous vs. synchronous mode for different batch sizes

The variance in synchronous mode is significantly higher than in asynchronous mode. Due to better latency in asynchronous mode, the energy consumption is lower than for synchronous mode. When looking at batch size 32, the latency per input is 18.8ms compared to 22.5ms with batch size 1 for sync mode. For async mode, there is even a better improvement noticeable from 20.6ms at batch size 1 to 14.9ms at batch size 32.

The most power-efficient mode is SyncB1, which needs approximately 6W of power. When an asynchronous mode is required batch size 1 is the best option. For latency, the best results are reached in asynchronous mode. Batch size 16 got the lowest latency. Other batch sizes greater than one are only a little bit slower. If optimized to energy, similar results can be seen. Asynchronous mode with batch size 16 is the most efficient. Synchronous energy is not that far behind, due to lower power consumption.

TABLE I. OPTIMIZED SETTINGS

Optima	CPU Cores	CPU Freq.	GPU Freq.	Async/Sync
Power	1	102MHz	76MHz	Sync B1
Latency	1-4	1.49GHz	921MHz	Async B>1
Energy	1	307MHz	537MHz - 921MHz	Async B>1

To summarize, we provide the optimum settings in Tab. 1. The reduction of active CPU cores to one can improve power and energy consumption while latency stays constant. CPU frequency can be reduced to a certain point where power, latency and energy is at an optimum. GPU frequency is the most important parameter for fast inference and it is also a good parameter to decrease power consumption when speed is less important. While asynchronous mode improves latency and energy, synchronous mode has the lowest power consumption.

D. Impact of Network Structure

Classification is a common use case for our platform. Therefore, we select the four classification networks MobileNetV1, MobileNetV2, ResNet18, and ResNet50. All of them were pre-trained for ImageNet classification. As an input, we use a tensor with a size of 3x224x224.

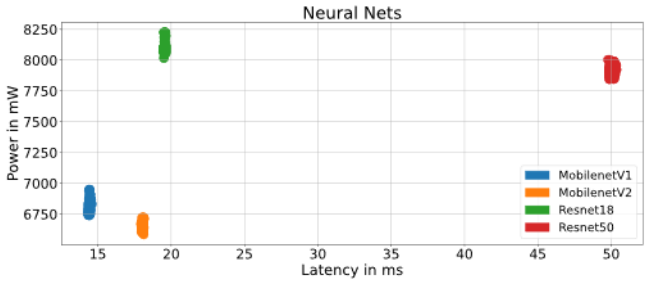

Figure 8. Power over latency of different Nets

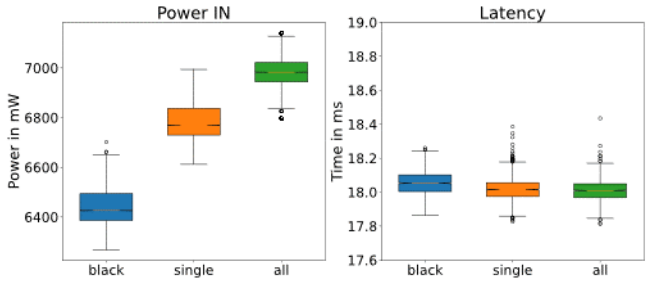

Figure 9. Image type effect on power consumption

In Fig. 8, the power per inference over the latency is displayed. Tab. II shows the accuracies of the networks. When comparing the MobileNets, we notice a slightly higher consumption in MobileNetV1. Furthermore, MobileNetV1 is 4ms faster than MobileNetV2. ResNet50 is the slowest, with a latency of 50ms. It is nearly 3 times slower than ResNet18. The power consumption of the two ResNets is similar. ResNet18 consumes a little bit more power than ResNet50.

In general, MobileNets are faster and more energy-efficient than ResNets at a comparable accuracy. The fastest inference and most energy-efficient classification can be achieved with MobileNetV1. When power is optimized, MobileNetV2 is 1% better than V1. Smaller ResNets seem to be outdated; not only do they need much more power, but also, the latency is higher compared to MobileNets.

TABLE II. OPTIMIZED SETTINGS

	MobileNet V1	**MobileNet V2**	**ResNet18**	**ResNet50**
Top-1 Accuracy	70.6%	72.0%	72.12%	77.15%

How power changes with different inputs can be seen in Fig. 9. We chose three test settings: a black image, a single random image from ImageNet, and 100 images from ImageNet. We collect power values from over 2000 inference cycles for every test case. After each inference cycle, we load a new or the same image into memory. We execute each of the 100 images 20 times. We execute the black and the single image 2000 times. While latency does not differ significantly, the input power increases 10% for a randomly picked image compared to only a black image. A fixed single random image increases power consumption by 5%. The cause for this might be that the black image results in simple operations with 0, which results in less flipping of transistors.

Then, we want to find out whether the power consumption of a layer changes when it is extracted from a network. A neural network is composed of individual layers. A question is whether a particular layer consumes the same amount of power in a single-layer network as in the original network. To answer that, we compose a small network that consists of the first layers of MobileNetV2 as a single layer network. To get comparable values, we analyze collected layer information from the parsed ONNX network. We look at every layer to see the input and output size as well as the layer type.

However, as TensorRT limits reverse engineering, we cannot get in-depth information like the kernel size of a convolution layer or type of activation. It is also not possible to delete layers to reduce the network to one layer of interest. Therefore, we extracted the missing layer information from TensorRTx Github page. Unlike the TensorRTx rebuild of MobileNetV2, RELU6, which is not supported by TensorRT, was supplemented with a standard RELU function.

In Fig. 10, we show the power and latency of the small network. To the left, the boxes contain mapped values of the layers. To the right, the boxes contain values from the single-layer networks. When comparing, we see a big power difference. Mapped values are much higher at 6W, while the single-layer values have a power consumption of about 4W to 5W. In general, the power consumption of layers embedded in a network does not match the power consumption of single-layer networks. Single-layer networks also have an initial layer, which is the input reformatter. This layer sometimes takes longer than the actual layer. Therefore, we take time values from the Nvidia profiler to get the values for the specific layer. While power values cannot be directly compared, latency values are comparable. The latency of the layers is equal in the entire network and single-layer networks. Energy values are also not the same due to the difference in power.

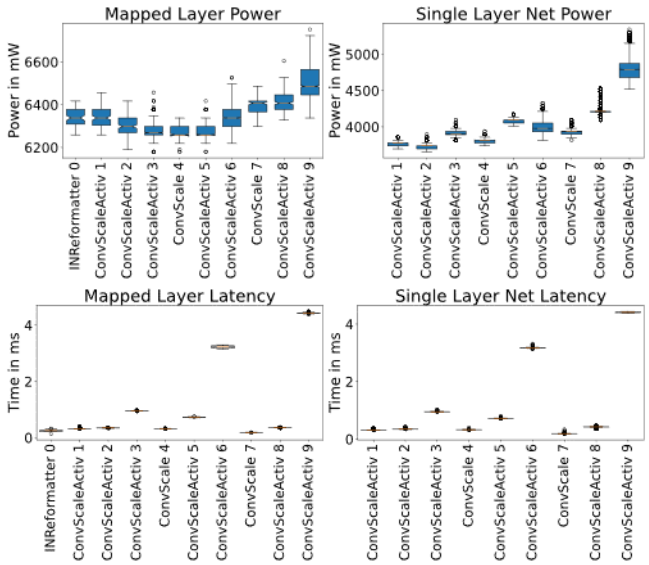

Figure 10. Power, latency and energy of single layers of MobileNetV2

A single-layer network can be used to compare latency but not to get the power consumption or the energy consumption of a layer in a network of more layers. The neighboring layers contribute significantly to power consumption.

Next, we provide an in-depth power analysis of the convolutional layer. We built and measured single-layer networks of a convolutional layer. We look at three parameters: (1) kernel size, (2) stride, and (3) output shape. The basic settings are kernel and stride to 1x1 and output and input shape to 3x224x224. The range in which kernel size and stride change are from 1x1 to 5x5. The output shape is set from 3x224x224 to 48x224x224, only change of one dimension. Padding is set so that no change in size occurs. Fig. 11 shows the effect on power consumption. Power increases with the kernel size, as already seen in the single-layer networks for MobileNet. A bigger kernel size results in more operations, therefore, more power. For stride, a decrease in power can be seen. Stride reduces the size of the output tensor; therefore, less power is needed. An increment in output shape causes an increment in power. Kernel size and the output shape have the most significant effect on power. When optimizing a network for power, a reduction of those parameters should be considered.

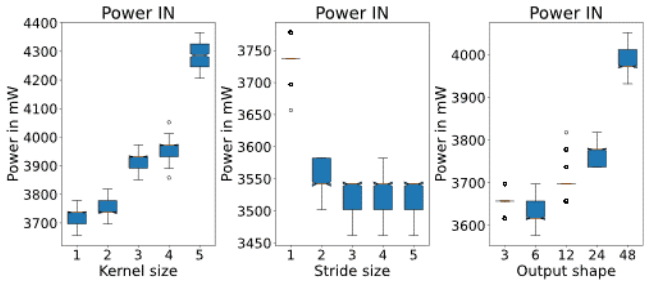

Figure 11. Power of different kernel, stride size and output shape of a convolutional network

To summarize, MobileNets are very energy efficient with comparable accuracy. Power consumption depends on the inputs: a black image needs less power than images with color. When using single-layer networks, latency is comparable with layers embedded in a network, but power differs significantly. Regarding the network structure, a smaller kernel in convolution layers, a stride setting greater than one and small output shape results in a lower power consumption

IV. Conclusion

We provided a reliable method to measure the power consumption of entire neural networks as well as single layers of a neural network on the NVIDIA Jetson Nano. The challenge was to get useful measurements with the limited time resolution of the onboard sensors, and we solved it by collecting data points from multiple inferences.

Measurements on four different hardware settings provide information on how inference can be optimized to power, latency, or energy. The settings are active CPU cores, frequency of CPU and GPU, and the effect of asynchronous mode. Optima have been found for MobileNetV2; for other networks, these optima can lie elsewhere. The results may also change when combining multiple parameters.

We tested how the network structure affects power consumption. Power differs from network to network, and it changes with different inputs. When rebuilding a layer of a network in a single-layer network, the power consumption is not the same as in the entire network. The neighboring layers do play an essential role in power consumption.

This work with accurate power measurements serves as a base for the development of power estimation models of the NVIDIA Jetson Nano. Further, we plan to extend our measurements with the same methods as in this paper to other related NVIDIA boards, such as TX2 and Xavier.

Acknowledgment

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology, and development are gratefully acknowledged.

References

[1] J. Kaster, J. Patrick, and H. S. Clouse. “Convolutional neural networks on small unmanned aerial systems,” in Aerospace and Electronics Conference (NAECON), 2017 IEEE National. IEEE, 2017, pp. 149–154.

[2] S. Luo, H. Lu, J. Xiao, Q. Yu, and Z. Zheng, “Robot detection and localization based on deep learning,” in Chinese Automation Congress (CAC), 2017, 2017, pp. 7091–7095.

[3] Mittal, S. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform. Journal of Systems Architecture, 97, 2019, pp. 428-442.

[4] E. Cai, Da-C. Juan, D. Stamoulis and D. Marculescu, “NeuralPower: Predict and Deploy Energy-Efficient Convolutional Neural Networks”, conference paper at ACML 2017

[5] Ž. Nakutis, “Embedded Sysems Power Consumption Measurement Methods Overview”, Kaunas Univerity of Technology, ISSN 1392-1223 MATAVIMAI. 2009. Nr. 2(44)

[6] C. F. Rodrigues, G. Riley and M. Luján, “SyNERGY: An energy measurement and prediction framework for Convolutional Neural Networks on Jetson TX1”, Int'l Conf. Par. and Dist. Proc. Tech. and Appl. | PDPTA'18 | 375

[7] C. F. Rodrigues, G. Riley and M. Luján, “Fine-Grained Energy Profiling for Deep Convolutional Neural Networks on the Jetson TX1”, 978-1-5386-1233-0/17/$31.00 ©2017 IEEE

[8] D. Kang, D. Kang, J. Kang, S. Yoo and S. Ha, „Joint Optimization of Speed, Accuracy, and Energy for Embedded Image Recognition Systems”, 978-3-9819263-0-9/DATE18/┐ c 2018 EDAA

[9] C. Xiao, G. Chen and W. G. H. Odendaal, “Overview of Power Loss Measurement Techniques in Power Electronics Systems”, IEEE TRANSACTIONS ON INDUSTRY APPLICATIONS, VOL. 43, NO. 3, MAY/JUNE 2007