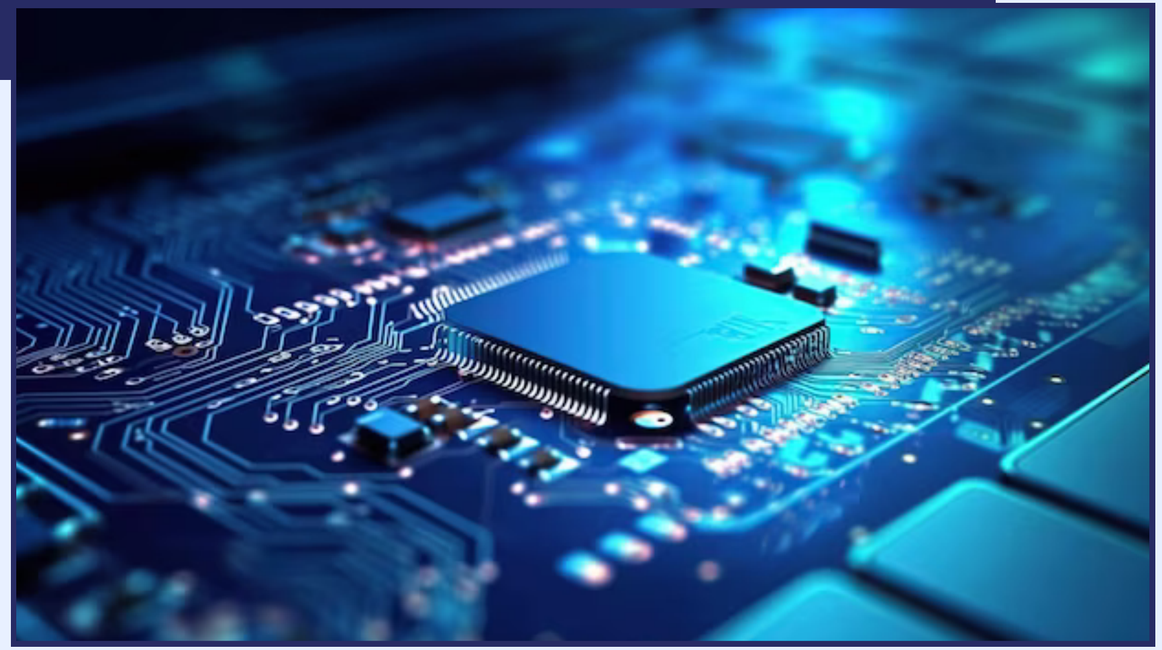


BENCHMARK ANALYSIS OF JETSON TX2, JETSON NANO AND RASPBERRY PI USING DEEP-CNN

Paper by [1]
Ahmet Ali Süzen
Burhan Duman
Betül Şen

Presentation by
Matteo Fragassi

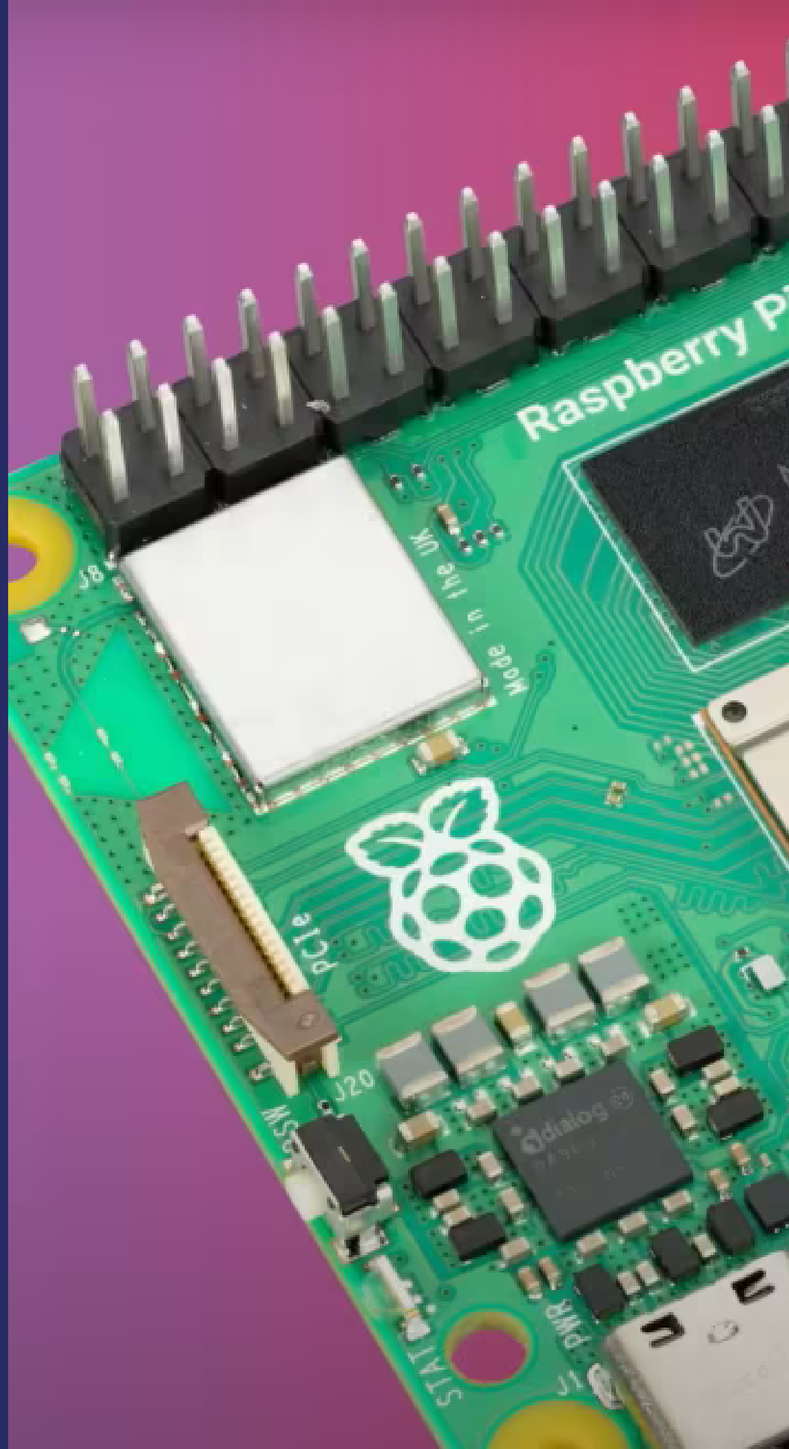
OUTLINE

- Background
- Machine learning challenges
- Experiment setup
- Results
- Summary
- Appendix

The objective of the paper is to benchmark three widely adopted single-board devices that can be used for deep learning applications

SINGLE-BOARD DEVICES & DEEP LEARNING

- **Single-board devices** are computers built on a single circuit board. They are simpler than general-purpose systems but, in most of the cases, they cannot be considered embedded devices due to their complexity. Popular examples are Raspberry Pi and Jetson platforms.
- **Deep learning** is a branch of machine learning that leverages neural networks with multiple hidden layers to progressively extract higher-level features from the raw input. It has different architectures such as Convolutional Neural Networks (CNN) and Transformers. Popular application fields are computer vision and natural language processing.



CHALLENGES

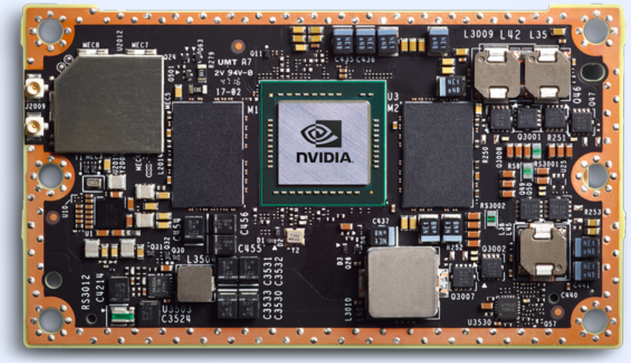
In some fields, ML has specific application scenarios, each with its own constraints

Automotive: autonomous vehicle	→	Power efficiency
Security: video surveillance	→	Reliability
Retail: analysis of customers behaviour	→	Cost
Safety: maintenance of critical infrastructures (e.g. bridges)	→	Prediction accuracy

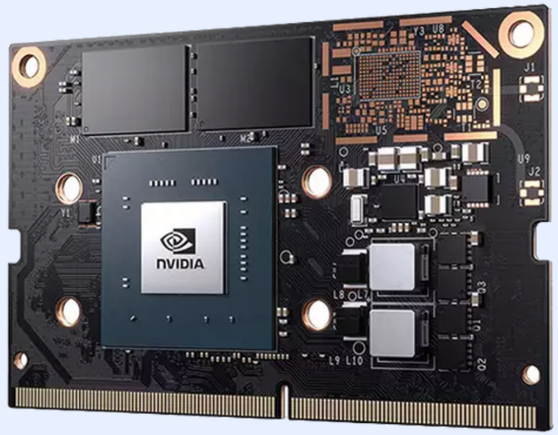
Speed is not always the main concern

SOLUTION

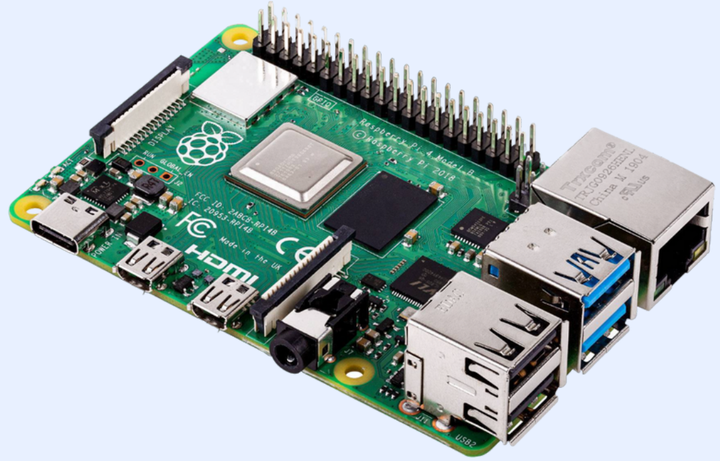
1. Analyze your problem
2. Select an ML model and a device to run it based on your needs



Jetson Tx2



Jetson Nano



Raspberry Pi 4

We will focus on the device choice

SETUP

Dataset

45k images divided in 13 classes from the DeepFashion2 clothing dataset [2].
5k, 10k, 20k, 30k and 45k subsets are used in different runs.

Evaluation metrics

- Training time / epoch
- Main memory occupation
- CPU and GPU power consumption
- Classification accuracy

Custom CNN model

Input (32,32,3)
Conv5: 32 5x5 kernels
ReLU (non-linear activation)
Max Pooling (2x2 window)
Conv5: 64 5x5 kernels
ReLU
Max Pooling (2x2 window)
FC layer (1024 hidden neurons)
Dropout (p=0.2)
Output FC layer (Softmax NL activation)

SETUP DETAILS

- The model is implemented in TensorFlow 1
- The benchmark is carried out during the model training on image classification.
- At each run the dataset is split so that 70% of the images are used for the training and the remaining 30% for the validation phase.
- Each training lasts 100 epochs.

BE CAREFUL !

Edge devices are usually evaluated on inferencing performances because classic architectures are too complex to be trained on them unless certain techniques are adopted (e.g. Transfer learning). Only simple models can be directly trained on single-board devices.

RESULTS

	Acc (%)			Time(sec)			Memory (GB)			CPU(Power/W)			GPU(Power/W)		
Dataset	**TX2**	**Nano**	**PI**	**TX2**	**Nano**	**PI**	**TX2**	**Nano**	**PI**	**TX2**	**Nano**	**PI**	**TX2**	**Nano**	**PI**
Idle	-	-	-	-	-	-	1,9	1,5	1,4	0,675	0,47	0,30	2,6	0,76	-
5K	87,6	87,5	87,2	23	32	173	2,6	2,0	2,1	2,23	1,50	3,5	5,27	2,23	-
10K	93,8	93,9	91,6	32	58	372	3,1	2,75	2,6	2,78	2,32	3,6	5,32	3,25	-
20K	94,6	94,5	-	52	-	462	4,5	ERR	4,0	3,76	-	3,9	5,22	-	-
30K	96,4	-	-	122	-	-	5,2	ERR	ERR	4,25	-	-	5,74	-	-
45K	97,8	-	-	235	-	-	6,5	ERR	ERR	4,92	-	-	6,29	-	-

ERR: memory error

TAKEAWAYS

- The larger the dataset the higher, the model **accuracy**

→ But more data also means more **training time**

- The higher the device performances, the higher the **power consumption**

→ However faster devices take less time so the **energy consumption** may be lower

- Big memories are necessary, especially during the **training phase**

→ The number of parameters grows with the model complexity

- **Cost** is also a key factor

→ Raspberry Pi 4: 35$
Jetson Nano: 89$
Jetson Tx2: 399$

MY RESULTS on Jetson Nano DevKit

I tried to replicate the experiment to check the results on the paper.
These are part of the monitored metrics.

Power consumption (mW)

Configuration	CPU	GPU	Total
Idle [MAXN]	500	0	2000
Idle [5W]	370	0	1800
5K dataset [MAXN]	1100-2700	2600	5800-7400
5K dataset [5W]	860	1400	4300

[MAXN=10W]

Memory occupation (GB)

IDLE	1.7
5K dataset	3.2
10K dataset	3.1

[Available memory = 3.9 GB]

Time (s/epoch)

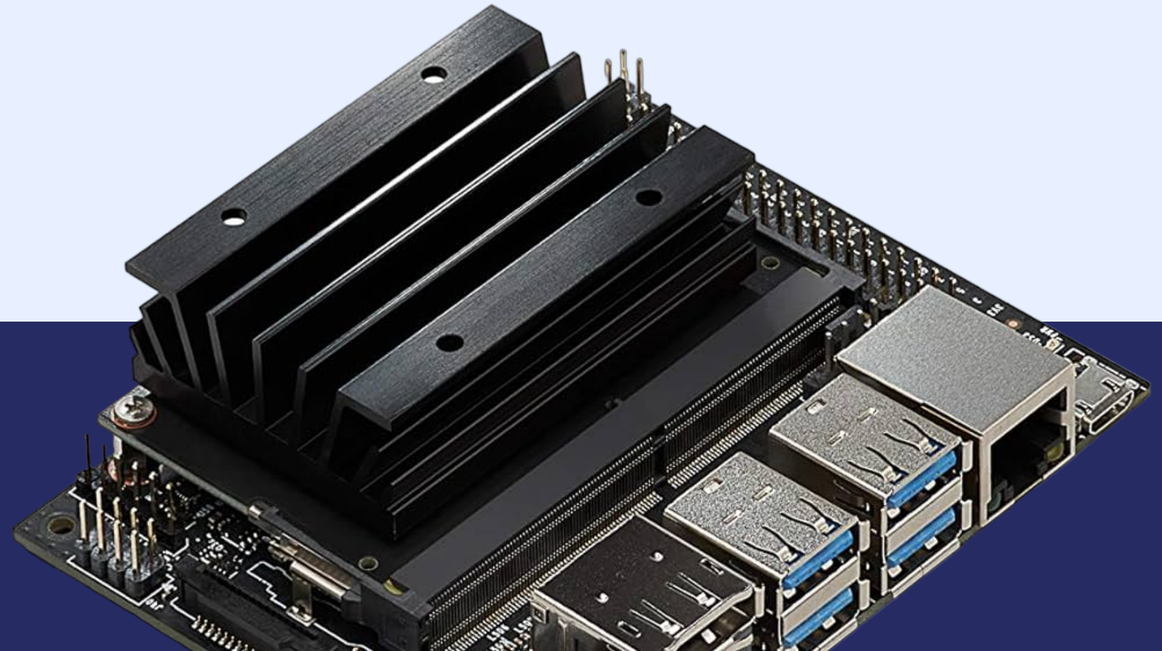
5K Batch32 5W	5K Batch32 MAXN	5K Batch64 MAXN	10K Batch32 MAXN
43	27	24	53

[BatchX means that there are X images per batch]

ANALYSIS

- The **power consumption** should not depend on the dataset dimension, unlike the paper results would suggest. Just to be sure I run a training session with 10K images and I got the same values reported for the 5K dataset.
- The network model doesn't change so the **memory occupation** should be about the same. Memory errors will always happen only if the batch is too large or the model has too many parameters.

Accuracy and other details are discussed in the appendix



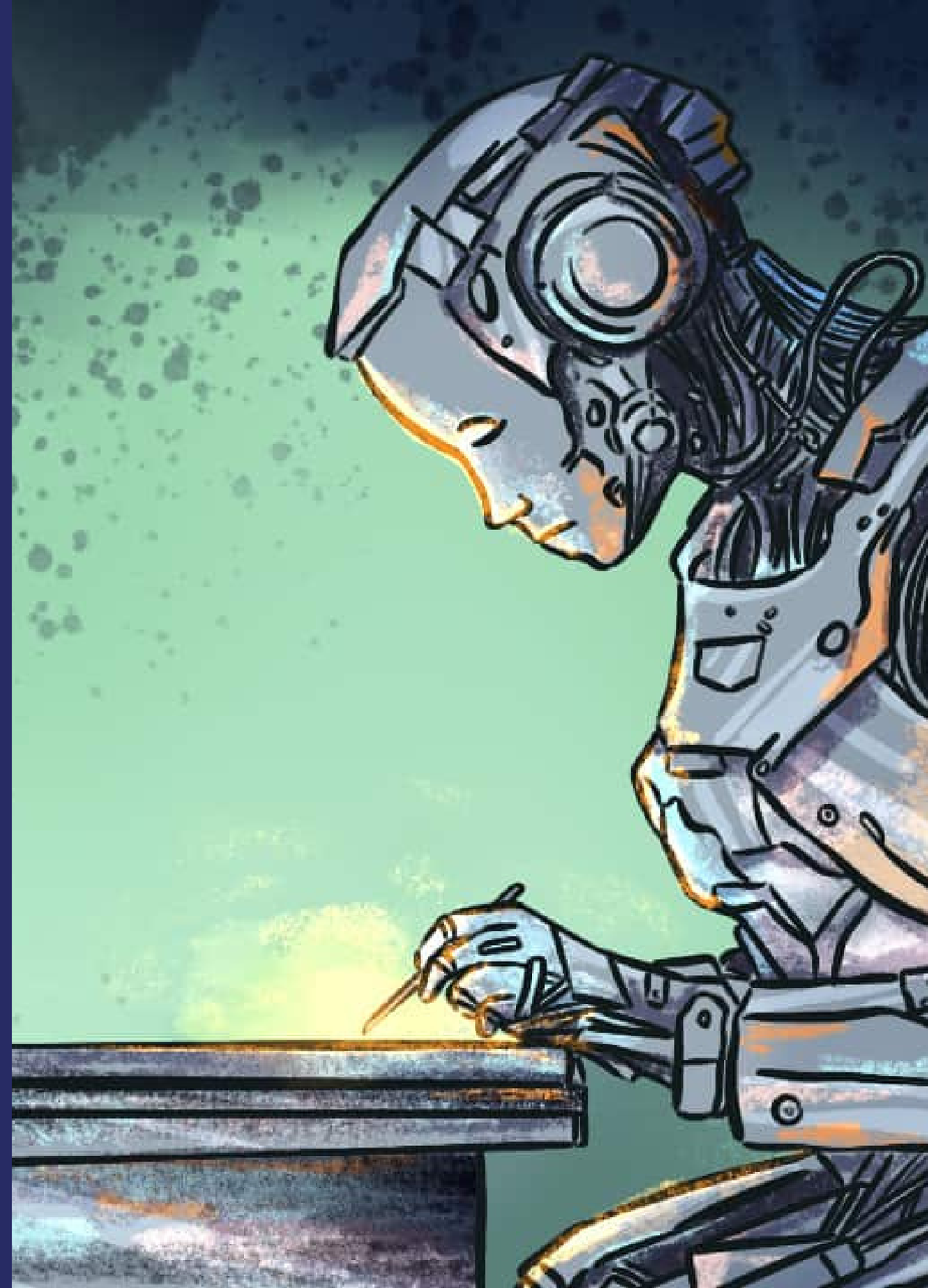
SUMMARY

Machine learning diffusion is inevitable, but the complexity of most algorithms requires powerful hardware to use them effectively. Single-board devices, especially if equipped with a GPU, can be a good solution to the problem because they balance cost, power consumption and performances.

THANK YOU

APPENDIX OUTLINE

- Boards specifications
- K-means
- My setup
- Accuracy
- Simpler datasets
- Final thoughts



BOARDS SPECIFICATIONS

The Raspberry Pi 4 comes with different memory sizes (1, 2, 4 and 8 GB). In this case the 8GB version was used.
The Jetson Nano and the Jetson Tx2 modules come in a single configuration. The developer kits version of these devices should be only used for developing purposes. The only difference between a Jetson module and a DevKit is the storage, since the latter has a micro-SD interface instead of an on-board memory [a1].

	Raspb. PI 4	Jetson Nano	Jetson TX2
Performance	13.5 GFLOPS	472 GFLOPS	1.3 TFLOPS
CPU	Quad-core ARM Cortex-A72 64-bit @ 1.5 GHz	Quad-Core ARM Cortex-A57 64-bit @ 1.42 GHz	Quad-Core ARM Cortex-A57 @ 2GHz + Dual-Core NVIDIA Denver2 @ 2GHz
GPU	Broadcom Video Core VI (32-bit)	NVIDIA Maxwell w/ 128 CUDA cores @ 921 MHz	NVIDIA Pascal 256 CUDA cores @ 1300MHz
Memory	8 GB LPDDR4	4 GB LPDDR4 @ 1600MHz, 25.6 GB/s	8GB 128-bit LPDDR4 @ 1866Mhz, 59.7 GB/s
Networking	Gigabit Ethernet / Wi-Fi 802.11ac	Gigabit Ethernet / M.2 Key E	Gigabit Ethernet, 802.11ac WLAN
Display	2x micro-HDMI *(up to 4Kp60)*	HDMI 2.0 and eDP 1.4	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4
USB	2x USB 3.0, 2x USB 2.0	4x USB 3.0, USB 2.0 Micro-B	USB 3.0 + USB 2.0
Other	40-pin GPIO	40-pin GPIO	40-pin GPIO
Video Encode	H264(1080p30)	H.264/H.265 (4Kp30)	H.264/H.265(4Kp60)
Video Decode	H.265(4Kp60) ,H.264(1080p60)	H.264/H.265 (4Kp60, 2x 4Kp30)	H.264/H.265 (4Kp60)
Camera	MIPI CSI port	MIPI CSI port	MIPI CSI port
Storage	Micro-SD	16 GB eMMC	32GB eMMC
Power under load	2.56W-7.30W	5W-10W	7.5W-15W
Price	$35	$89	$399

K-MEANS

“k-means clustering is a method of vector quantization that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.” Wikipedia

Usage in the paper

In the paper the K-means algorithm is used to preprocess the data in 21 different classes. The reclassified data are then recombined and used to form the training datasets.

Dataset problems

After the K-means, the data are divided in 21 classes while the CNN classifies the inputs in 13 classes. The authors do not mention how the data have been processed before using them as inputs of the network. This means that we do not really know the used dataset.
It is also possible that the CNN model description is wrong and the output should be on 21 classes. However, the problem is still present since the K-means output depends on the centroids initialization.

MY SETUP [1]

Board

The experiment was carried out using a Jetson Nano Developer Kit with JetPack 4.6.4 and python 3.6.9. The utility used to monitor power and memory occupation is Jetson-stats [a2] while the Tensorflow 2 package provided by Nvidia can be installed following the guide at: https://forums.developer.nvidia.com/t/official-tensorflow-for-jetson-nano/71770

CNN

The CNN architecture lacks some information to be reproduced exactly in Tensorflow. For example the kernels stride and the convolutional layers output padding are not specified. These information can normally be omitted since benchmarks are performed with well know architectures, but this is not the case.
Another important detail is that the convolutional layers are reported with the label “Conv5”. This term is commonly used to identify a block of the VGG16 architecture so I implemented it like in the said architecture.

MY SETUP [2]

Dataset

The model has been trained using the DeepFashion2 dataset together with other datasets already provided by Tensorflow (e.g. Cifar10, Mnist, Fashion Mnist, etc).
The DF2 dataset has been preprocessed in different ways since the exact composition of the dataset used in the paper is not known.

Preprocessing steps

- If an image has more than one object, the single objects need to be extracted since our model is not able to perform multiple classifications
- Normalize and center the data

Data augmentation was not performed because the unbalanced dataset is not a primary issue.

ACCURACY

The **validation accuracy** obtained with a 10K dataset is not even close to the one declared in the paper.
Increasing the data does not increase significantly the accuracy.
The results are the same if we change the **Conv5 block** with a single convolutional layer or with the same block of the VGG19 architecture.

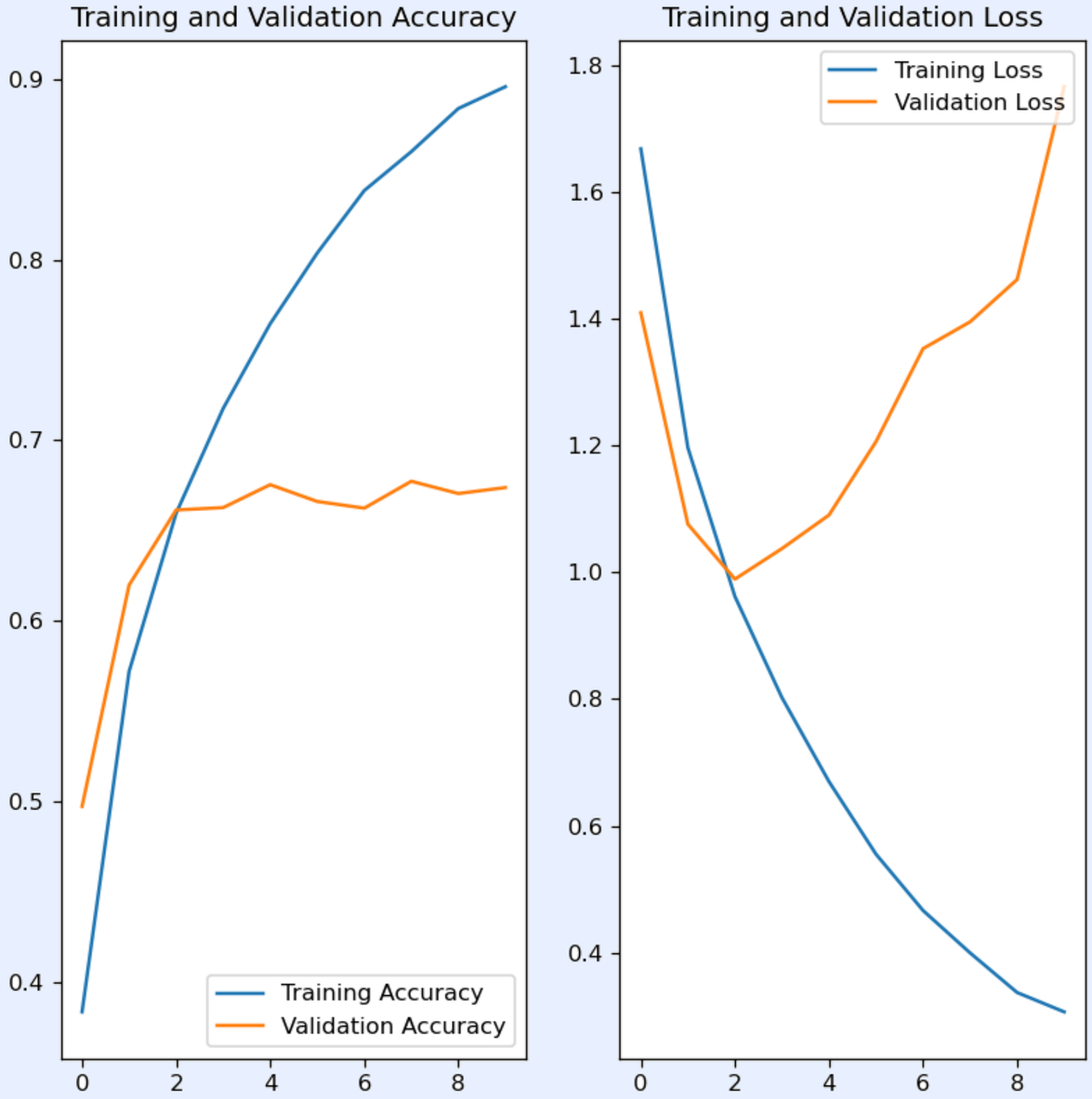
The **batch dimension** was not specified in the paper, however, by looking at the time results [a3], it seems reasonable to assume that there are 32 images per batch.

10K dataset - batch 32 images - 25 epochs

SIMPLER DATASET

A possible explanation to the low accuracy, supposing there are no error in the architecture description, could be the **poor learning ability** of our model. For this purpose the Fashion MNIST and the Cifar10 datasets have been used.

The reached accuracy is a poor result given the simpleness of the dataset. Even a less complex model could get better results than these [a2].



Cifar10 - batch 32 images - 10 epochs

FINAL THOUGHTS

I was not able to replicate the paper results due to a lack of information on the experiment setup. This does not mean that the results are not valid since there are many scenarios I didn't consider. For example the K-means, with K=21, may have simplified so much the data that even a simple model could have achieved high accuracy values. In fact our architecture could reach good results with the MNIST dataset (99% accuracy), but so can a shallow network.

For more information you can check the provided code.

THANK YOU FOR THE ATTENTION!