DOI: https://doi.org/10.46354/i3m.2019.emss.016

SOFTWARE FRAMEWORKS FOR ARTIFICIAL INTELLIGENCE: COMPARSION OF LOW-LEVEL AND HIGH-LEVEL APPROACHES

Michael Bogner[(a)], Florian Weindl[(b)], Franz Wiesinger[(c)]

[(a), (b), (c)] University of Applied Sciences Upper Austria – Department of Embedded Systems Engineering
Softwarepark 11, A-4232 Hagenberg, Austria

[(a)]michael.bogner@gh-hagenberg.at, [(b)]florian.weindl@fh-hagenberg.at, [(c)] franz.wiesinger@fh-hagenberg.at

ABSTRACT
As nearly every artificial intelligence application is based on a framework, using the best fitting one for the task is key in developing an efficient solution quickly. Since there are two main types of frameworks, based on low and high abstraction level approaches, these two types will get compared and evaluated throughout this paper using Tensorflow and Keras as representatives. Key features of artificial intelligence frameworks for industrial applications are performance, expandability, abstraction level and therefore ease of use for rapid prototyping. All those features are major factors to keep development time and costs as low as possible, while maximizing product quality. To evaluate both approaches by these criteria a neural network classifying handwritten digits is implemented.

Keywords: tensorflow, keras, neural network, evaluation

1. INTRODUCTION

Since the middle of the 20th century scientists have been trying to implement forms of artificial intelligence on computer systems. As time went by, those systems have grown from small programs to enormous applications. For each new application of artificial intelligence (AI) a specific neural network was designed and implemented, specifically tailored to the needs of the application. This had to be done since there had not been enough computing power and memory available to develop more general solutions.
Since implementing all basic functionalities each time from scratch is very time consuming and huge amounts of computing power became available year after year, artificial intelligence frameworks have been developed. These frameworks pack useful functionality and a basic environment into a reusable package, making development of bigger and better programs quicker and more convenient. Today nearly all AI applications are based on such software frameworks enabling high flexibility and performance all while keeping development time down to a minimum. Most of the time these generally very basic, reusable software packages are customizable to fit very specific and demanding tasks.
Since artificial intelligence and neural networks are widely used in many commercial products and industrial applications, the number of frameworks being available keeps growing. Based on this fact it is not that easy to choose a fitting framework for a specific task or product. If the wrong software is used, many problems may arise. Those problems may range from slower development and consequently to a longer time to market, to a complete project failure caused by major performance hits.
Some of these artificial intelligence frameworks stand out from the crowd because they are based on a very efficient concept or they are backed and continuously developed by concerns such as Google.
The goal of this paper is to compare the two basic concepts of artificial intelligence frameworks: high- and low-level approaches. Frameworks using a high-level approach do not clutter function interfaces and application programming interfaces (API) with unneeded parameters and details. By using such an approach implementation is done based on more abstract functionality and system blocks, which makes development fast and easier to begin with. Low-level frameworks on the other hand tend to give a very detailed and powerful API, enabling the developers and engineers to tweak and optimize every little setting and element of every component of the complete system. This ability enforces great flexibility and efficient applications but comes with costs of longer development times and more needed know-how of all processes in detail.
As representatives for high- and low-level approaches Tensorflow (Google Inc. 2019) and Keras (Keras Home 2019) have been selected, since both are very widely spread in the research community and in industrial environments as well.
Theano (Theano 2019) would have also been an available option as low-level framework. Based on the NumPy (NumPy 2019) library the software is able to translate all operations into efficient C programming language, enabling high performance.
Another solid choice for a high abstraction level framework would have been Caffe (Berkeley AI Research 2019) due to efficient grafics processing units (GPU) usage and abstract modularization.

The fact that Keras, as high-level framework, is based on Tensorflow ensures the opportunity to compare both frameworks against each other and is the main reason their selection.
For evaluating the performance, the abstraction levels and rapid-prototyping ability a neural network gets implemented using both frameworks and classifying handwritten digits of the MNIST dataset (LeCun Y.L. 1998). This prototype network will consist of 8 layers, transforming the 28- by 28-pixel sized images on its ways through the network and at the end predicting the result.

2. BASICS OF ARTIFICIAL INTELLIGENCE

The main goal of the specialist division of artificial engineers and scientists is to create a system which is capable of making decisions like a real human being would do. To keep the complexity of such systems at a still computable and achievable level an AI or neural network is engineered for one single, very specific application only. Such applications may be daily things such as "face-unlock" on smartphones or customer analytics on online platforms, like amazon.
To enable a computer system to perform such complex tasks they need to be taught and trained. This training phase is a mandatory step and can be done in a few different approaches. In this case supervised learning (Schmidhuber 2015) will be used, where all training data is connotated with the correct answer the system should give. The AI is fed with this dataset repeatedly until it is able to discover different patterns in the data to predict the correct answers for the given information. After each so-called batch, which represents a small subset of the training data, the network is given feedback on its answers, providing possibilities to change its internal parameters to improve its accuracy. Because the system is supervised and feedback is given throughout the complete training process, this method is called supervised learning.
After the network is trained to a sufficient level, which is determined by an error function or error rate, the AI is ready to be deployed on the final product. Training in the beginning plus setting up all different parameters and the datasets is quite time consuming and is therefore mostly done on specific hardware to accelerate these processes. If the network is smaller in most cases the training is done by one graphics processing unit (GPU). For larger scale applications the training phase is done on multiple GPUs or a server farm, sometimes even on cloud servers.

2.1. Neuronal Nets

In case of this prototype where handwritten digits will be classified, image patterns need to be found in the data. For this specific task neural networks (Lunze 2016) provide a perfect fit. Such networks consist of single neurons that try to mimic the functionality of real neurons in the human brain, both give a specific output, if its input connections are stimulated in a certain way. In case of a simulated neuron of a neuronal network these inputs and outputs are not limited in quantity, enabling the neuron to be connected to one or more neurons in its environment. Based on a specific mathematical function like sigmoid, also called the activation function of a neuron, the output is set in a specific way (Gershenson 2003).
The connections between different neurons transporting data from one neuron to another through the network are called edges. These edges may be simply used for transport between neurons, but most certainly will also introduce some weights on the data. By a multiplication with a variable factor the edge values are altered and therefore able to inhibit or constrain the transport of data to specific neurons. By varying those weights of every edge in the network the AI is adapting to given data and optimizing its error rate to a minimum and hence learning (Schmidhuber 2015).
To keep all neurons in a logical and clear order, a neural network consist of different layers, each representing a single functional group of neurons doing simple operations on the given data. A simple network for image recognition may for example involve an input layer, containing one neuron per image pixel, of many intermediate or hidden layers and an output layer, where each neuron is representing one possible outcome of the network. As the tasks of a network get more complex its layer count increases as well as the neuron count per layer, which quickly bursts the limit of network complexity and demonstrates why an AI should be developed for one single application.

2.2. Convolution

A specific type of neural network used in image recognition is called convolutional neural network (LeCun 1995), which uses the concept of convolution to extract features out of the given pictures.
Convolution is done by sliding a filter of a given size over the whole image and calculating the mean value of all pixels within the filter. The calculated value is then set as new pixel value for the next layer. This convolution layer is always paired with a pooling layer, summing up a small area around a pixel or just taking the maximum value and therefore reducing the size of the image. By convoluting and decreasing the image´s size, features and patterns get picked up by the network and overall performance is increased. The combination of a convoluting and a pooling layer is very common in image recognition and is often used more than once to decrease data size further and extract features.

3. METHODOLOGICAL APPROACH

In order to be able to objectively compare high- and low-level frameworks to each other to representatives have been chosen. Both frameworks are based on the same backend software which in this case is Tensorflow.
To compare performance, complexity, usability and rapid-prototyping abilities a prototype convolutional neural network gets implemented using both frameworks. The goal of this prototype is to be

demanding enough to show differences in compute performance but at the same time keep training times to a minimum and get a decent insight in working with the frameworks. For this task classification of handwritten digits of the MNIST dataset was selected since working with 28- by 28-pixel images in multiple layers requires a fair bit of computation power and will show possible bottlenecks or high optimized parts within the framework.
After the implementation the prototypes of both frameworks will get trained and tested on the same datasets to keep errors in accuracy to a minimum.

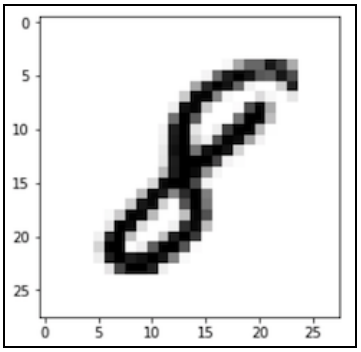

Figure 1: A handwritten digit of the MNIST dataset, representing the digit eight as used in the prototype (Gazi Yalcin O. 2018).

4. PROTOTYPE

As representative for high level frameworks Keras was chosen for its high abstraction level and its current spread on AI topics. On the other hand, Tensorflow was selected as low-level representative based on the high scalability and the opportunity to customize and adapt nearly every process within the framework. Tensorflow may also easily be used as base for Keras to build on, so both frameworks can be objectively compared using the same basic software, regarding data processing and control flow. Both frameworks are supported by big and active communities and are documented very well online making them very attractive to potential users. Also, it is unlikely for them to stop getting frequent support from its developers.

4.1. Concept

To evaluate all pros and cons of both frameworks a prototype network is implemented using both. This convolutional neural network will be developed to classify handwritten digits. The internal structure of the network itself will be explained in detail in the next paragraph. To train both networks to a comparable level the Modified National Institute of Standards and Technology (MNIST) dataset (LeCun Y.L. 1998) will be applied, containing 60,000 images for training and 10,000 images of digits for evaluating the accuracy. All these images were created by 500 different people, offering plenty variation in the dataset to prevent the network from overtraining on special features of unique handwritings. An example for such a handwritten digit may be seen in Figure 1.
This prototype was chosen on the premises of being complex enough to use advanced features of both frameworks and provide long enough training phases to compare the efficiency of both environments. Yet it is quite simple to implement the network with a few basic layers commonly used in many applications working in the same basic principle in both Tensorflow and Keras.

4.2. Network

As mentioned before the best network for such an image recognition task is a convolutional neural network, in this case with eight layers and two stacked cascades of convolution and pooling layers. Figure 2 shows the structure and dimensions of this network, all noted numbers are indications for used neurons in these layers.
At the input layer a 28x28 pixel greyscale image of a handwritten digit gets fed into the network, where the information of each pixel is taken by one neuron. The first cascade of convolution and pooling filter takes the original image, convolutes and resizes it to a dimension of 14 pixels squared. This process is repeated in the next cascade, which resizes the data to 7 pixels squared. This method greatly reduces data inside the network and keeps the number of neurons down, therefore increasing the system's performance in training by a significant amount.
This step is followed by two fully connected layers, in which every neuron is linked to every other neuron within the layer. Such layers are very good in combining all the features detected by predecessor layers. They associate multiple inputs and try to predict the correct digit based on the detected features. Since the data is coming out from the last layer in a 7x7 format with 64 channels or features for the last pooling layer, this layer consists of a great number of neurons. To further optimize and increase accuracy of network predictions, another fully connected layer, this time with 1000 neurons, is added onto the back. The last needed part of the network is an output layer, which consist of the amount of prediction outcomes possible. In this case this layer features ten neurons because there are ten digits from 0 to 9 for the network to predict. At the end each neuron shows an output value from 0 to 1, representing the probability for the picture to be this specific number.

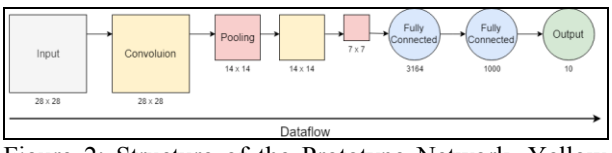

Figure 2: Structure of the Prototype Network, Yellow and Red Layers Represent Convolution and Pooling Layers Respectively.

4.3. Evaluation Criteria

To evaluate both types of frameworks based on objective criteria, they are tested on the following four

aspects: Performance or accuracy, rapid-prototyping capability, expandability and training performance. All of these properties are crucial for industrial applications. The most critical point of a neural network is its ability to make accurate decisions and predictions, therefore the performance of the network and its underlying framework is key. Also, very important is the ability of a framework to allow its users to create rapid prototypes for the first evaluation of ideas or new concepts. This also saves a lot of development time and will be mainly evaluated based on the ease of use when implementing the prototypes. This feature makes great difference in project costs, since faster development time directly correlate to less financial effort for companies. Often frameworks do not deliver all needed functionality out of the box, this is where possible expandability of such systems comes into play. If the environment is easily adapted to new functionality and new concepts, much effort can be saved in this stage. The last important criterion is training performance and the ability to upscale the training. Since each neural network nowadays is trained on a GPU, easy and efficient support for such hardware units plays a huge role in an overall performance of the network. This gets tested by training the network on a GPU, since the process as well as the code is nearly the same as for training on specified Tensorflow Processing Units (TPU), explicitly made for training AI. This opens the possibility to further improve AI performance, while still using the same training time, or cutting the training effort and keep the quality of the final network. These options are highly beneficial for commercial or industrial applications.

5. TENSORFLOW

Tensorflow was and still is developed by the Google Brain Team. It is specified in data processing on heterogenic systems, enabling the framework to be very efficient on large multi-core processing units, GPUs und also Google's own Application Specific Integrated Circuit (ASIC) device called TPU. Those TPUs have been solely created to accelerate the training process of neural networks and AI in general (Abadi 2016).

The unique features of Tensorflow are the data flow graph and the so-called tensors. The Dataflow graph is a directed graph, which describes the proceeding of data through the network. Each node of this graph is a representative for a layer in the prototype network and abstracts one or more operations on the data. The term tensor in this case describes an n-dimensional data field, may being one of Tensorflow's base datatypes, such as *int32*, *float32* or *string*. A tensor is always used to hold data between nodes in the graph, there for a node needs at least an input- and output-tensor. The filling state or dimensions of such tensors may vary from operation to operation, since not every operation is based on the same dimensions (Abadi 2016).

A matrix-multiplication of a polling node may, for example, change the dimension of the data based on its input sizes. Such a graph is shown in Figure 3. This graph is automatically generated by the framework and represents the frameworks internal structure of the prototype.

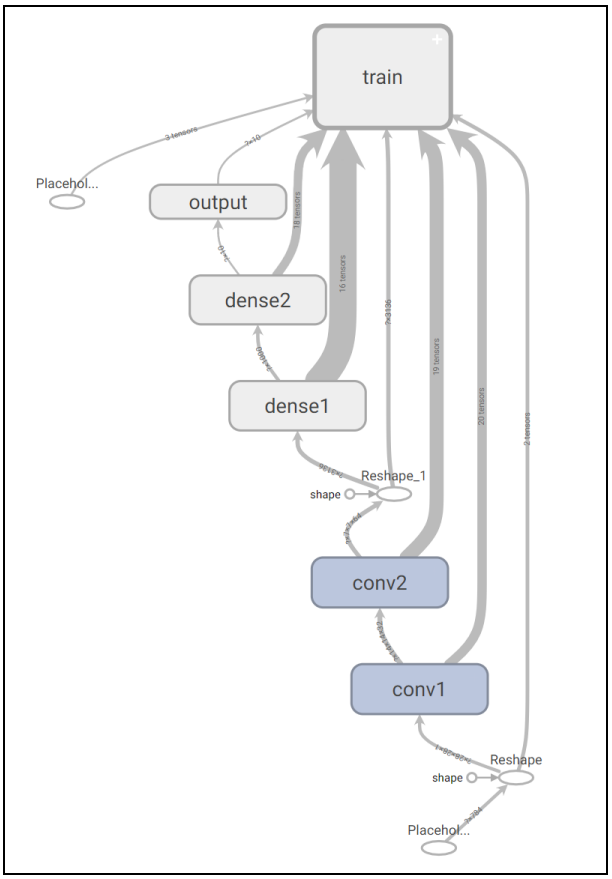

Figure 3: Tensorflow Graph of the Prototype

The major upside of Tensorflow is the ability to scale the systems very efficiently based on hardware acceleration. It is easy to execute parts of the dataflow graph on a GPU or TPU. If needed it is even possible to process the whole graph onto one of these devices. Since Tensorflow is based on a very low abstraction level, every little function can be tweaked and adjusted, which results in an immensely high optimizing potential and if you are willing to spend some time, also in very efficient and nearly infinitely scalable applications. However low abstraction levels are not only positive, they are also Tensorflow's greatest downside. Since rapid-prototyping requires a fast and easy to use environment, it is quite challenging to create a quick prototype of a desired network. Each function needs a specific amount of information about its data and parameters. Those values need to be set in order to achieve a decent accuracy of the network or make it work in the first place. So, a lot of knowledge and a long adjustment period is needed to get the most out of Tensorflow networks. Using Tensorflow is therefore prolonging time to market and development cost by big amounts.

Setting up the environment to use Tensorflow is not challenging and very well documented on the webpage. After installing all needed libraries for GPU support, it

is a matter of changing a few lines of code to get Tensorflow to also use available GPUs or TPUs. Tensorflow features a steep learning curve and needs some time to get used to and be productive with the environment. Once you are comfortable with the API there are a lot of useful tools like the included visualizer called TensorBoard. This tool is a great way to double check the correct layout of your neural network and also visually see the ongoing training stats like accuracy and training time in charts.

6. KERAS

Keras is an AI framework for Deep Learning developed by Francois Chollet in 2015. It features a high abstraction level and is available for Open-Source use. Since Keras only is a Python library that needs to be based on a backend software, it can be paired with a few different low-level systems such as Tensorflow, CNTK or Theano. The goal of Keras is to abstract from complex API functions and tons of different tweakable parameters and offer the opportunity to quickly develop a quite powerful neural network out of a few components (Keras 2019). With this approach it is easy to implement first prototypes or react to fast changing specifications. It is also well documented and does not need a lot of special know-how in neural networks to get started with development. Since Keras is based on a low-level framework it inherits much of the performance benefits of its base framework. Due to not having the opportunity to solve all problems in high-level code tuning, some parameters may be necessary at one point. It is possible to write code directly in base-level framework syntax and abstraction to further optimize (Keras 2019).

Because Keras in this case is based on Tensorflow it also uses its dataflow graph mechanisms and therefore generates a visual representation of the internal network, which can be seen in Figure 4. In this figure the 'train' node of Figure 3 is decomposed in its subfunctions 'metrics' and 'loss' shown in the top of the graphic.

The framework and the workflow itself was easy and fast to set up once all required libraries and the background framework was installed. Since Keras is a high-level framework with great abstraction getting used to working with the environment and different abstract layers was fast. An early prototype of the network could be developed within a few hours using the 'Sequential Model' offered by the framework, where layers only get added to one another in a sequential fashion. All functionality not available in basic layers and functions may be added or customized in any way. This however needs to be done in the abstraction level of the basic framework, which in this case requires the developer to be able the use Tensorflow as well as Keras. The documentation of the framework is based on very detailed descriptions of all used parameters and settings. This took a while since there are no real examples on how to correctly use the given API functions. But since there is a big community behind Keras it was possible to find all needed information online.

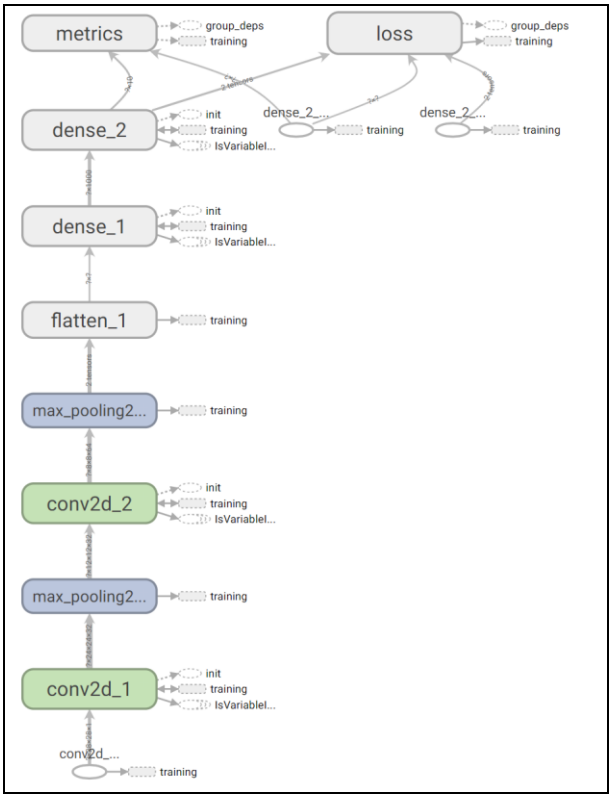

Figure 4: Graph of the Keras Network, Created by Tensorflow Backend Software.

7. TESTING ENVIRONMENT

As testing environment, a computer with a Windows 10 operating system was used. The system is based on an Intel i7 5820k Processor, 16 gigabytes of DDR4 RAM and a Nvidia GTX1080Ti GPU.

For the frameworks a Python environment was needed, which was used in version 3.6.2 and 64 bit. This enabled the test to use Tensorflow version 1.7.0 and Keras version 2.2.0. To test both training performances on CPU and GPU libraries from Nvidia were used. For CUDA support version 9.0.176 was paired with the neural network library cuDNN version 9.0 from Nvidia. At the time of testing an evaluation all the above listed pieces of software had been the latest stable versions.

7.1. Testing Procedure

Both networks were trained on the CPU of the system as well as on the GPU of the system. Each test was conducted 5 times to rule out variances in accuracy or training speed. All results shown further on are averages over these 5 test runs, which were completed with a batch size of 128 images per batch and 20 revolutions of the dataset. One single iteration over all the batches of the dataset is called an epoch further on.

8. TEST RESULTS

All conducted tests have shown that the combination of the network, optimizer and dataset would have also

been nearly the same if only half of the epochs had been used for training. After the 10th epoch only minor improvements can be seen in accuracy. In most cases such minor increases in accuracy are not worth extra training time, since accuracy is at this point already at about 98 to 99 percent. Accuracy in this case directly relates to correctly classified digits. However, Keras reaches its final accuracy level a bit faster than the Tensorflow network, which gives the possibility in training even less while reaching the same quality.

8.1. Tensorflow

As shown in Table 1, Tensorflow reaches average accuracy of 0.9880 or 98.80 percent by training the network on the system's CPU, which takes about 16 minutes of training time.

In contrast, using hardware acceleration from the system's GPU an average accuracy of 0.9895 was reached. This took around 40.2 seconds and therefore drastically improves efficiency. By shortening training times more development may be done in the same amount of time, which results in well-defined networks being able to deliver better quality results. Which means the GPU training recognizes digits with an equal precision but does this approximately 25 times faster.

Table 1: Results for Training on CPU and GPU using Tensorflow, all Time Measurements are noted in Seconds.

GPU		CPU		Test number
Accuracy	Time	Accuracy	Time	
0.9896	40	0.9893	980	1
0.9904	40	0.9861	977	2
0.9887	40	0.9890	974	3
0.9899	41	0.9887	970	4
0.9891	40	0.9868	975	5

8.2. Keras

As shown in Table 2 Keras reaches an average accuracy of 0.9989 on the CPU, which takes around 510 seconds. With this accuracy only about 1 in 1000 digits is detected wrong. By training the network on the GPU the same accuracy can be reached. The only difference in this case is the reduced training time needed to 53 seconds on average. In all test runs Keras reached its peak accuracy level a few epochs faster than Tensorflow did, whereas overall training times on GPU based training was longer.

8.3. Comparison of results

By looking at the results one can quickly see the performance and scalability benefit of Tensorflow, being about 31 percent faster when also using the system's available GPU to process data. By looking at the CPU results at first glance there seems to be an error in testing due to the longer training time needed by the more performant framework Tensorflow. But this difference in time is caused by not tuning every little parameter to the best possible value, which represents using the framework in a way an engineer would do when development time is limited and there is no chance to tweak and optimize every little setting. Additionally the Tensorflow code consists mainly of self-implementations, where by using Keras everything for this prototype needed is already provided by the framework environment. This approach also highlights the difference in system modelling between both frameworks. Tensorflow all task and groups need to be modelled quite detailed, whereas Keras enforces a more abstract modelling approach by only needing very abstract process descriptions. Both representatives were used as they would have been in a more complex and bigger industrial environment, to keep objective comparison possible.

Table 2: Results for Training on CPU and GPU using Keras, all Time Measurements are noted in Seconds.

GPU		CPU		Test number
Accuracy	Time	Accuracy	Time	
0.9988	53	0.9986	502	1
0.9994	52	0.9986	506	2
0.9985	54	0.9993	523	3
0.9992	54	0.9993	526	4
0.9988	52	0.9986	525	5

9. CONCLUSION

Although the test series showed performance benefits when using Tensorflow in combination with hardware accelerators, the choice of best framework depends on the intended application and needed key features. If data amounts and complexity stay within average, the high abstraction level of Keras is far superior over the high optimizing potential of Tensorflow. Giving the user the opportunity to develop faster prototypes and test more revisions improves the quality of the final network way more than modelling every single function in detail.

If Keras is then also based on Tensorflow as backend software, they form a great symbiosis where weaknesses of both get covered by the other framework. If an operation is missing in the Keras framework, it is possible to add this functionality in highly optimized Tensorflow native code and keep the rest of the application clearly arranged in high-level code, therefore having the benefit of fast development times combined with great performance.

These features make Keras, and high-level frameworks in general more suited for small to medium sized tasks, where rapid-prototyping is essential and development time advantages weigh more than pure performance and customizability. Therefore, Keras is better in keeping the costs down in development while still delivering good quality results. Although expandability is not as simple as with Tensorflow, all other key features of an efficient AI framework for medium sized projects are covered by the Keras framework, making it great for embedded projects like facial recognition on smartphones or interpreting telemetry data of robots and machines.

Large systems, like customer analyzing, however, suffer performance hits caused by the abstraction level and overhead of Keras where Tensorflow on the other hand is the perfect choice. Due to highly specific and custom modelled processes working with huge amounts of data, Tensorflow is able to deliver the required performance. However, to achieve these results substantial expertise and AI know-how must already be present within the development team. Based on the evaluated key criteria, low-level frameworks like Tensorflow tend to be more useful in scenarios where huge amounts of sensor or image data need to be processed. One example of such an extremely demanding application would be autonomous driving of vehicles, where lots of sensors need to be checked thousand times per second and correct decisions need to be made near instantly. To model such networks the fine granularity and the optimization potential of Tensorflow and low-level frameworks in general is essential in developing efficient and powerful products and keep development effort and therefore product costs as low as possible.

10. REFERENCES

Gershenson C.G., 2003. Artificial Neural Networks for Beginners. Eprint: cs/0308031.

Schmidhuber J, 2015. Deep learning in neural networks: An overview. Available from: http://www.sciencedirect.com/science/article/pii/S0893608014002135 [accessed 10.07.2018].

Lunze J, 2016. Künstliche Intelligenz für Ingenieure: Methoden zur Lösung ingenieurtechnischer Probleme mit Hilfe von Regeln, logischen Formeln und Bayesnetzen. Available from: https://books.google.at/books?id=NqBlCwAAQBAJ [accessed 11.08.2018].

LeCun Y.L., Cortes C.C., Burges C.J.C.B., 1998. THE MNIST DATABASE of handwritten digits. Available from: http://yann.lecun.com/exdb/mnist/ [accessed 13.07.2019].

Abadi M.A., 2016.Tensorflow: A system for large-scale machine learning. Available from: http://arxiv.org/abs/1605.08695 [accessed 06.07.2018].

Keras Home, 2019. Keras Home. Available from: https://keras.io/#keras-the-python-deep-learning-library [accessed 02.08.2018].

Keras, 2019. Keras Why use Keras. Available from: https://keras.io/#why-this-name-keras [accessed 02.08.2018].

Google Inc., 2019. Tensorflow Homepage. Available from: https://www.tensorflow.org/ [accessed 11.07.2019].

LeCun Y.L., Bengio Y., 1995. The Handbook of Brain Theory and Neural Networks.

Theano, 2019. Theano Homepage. Available from: http://deeplearning.net/software/theano/ [accessed 11.07.2019].

Berkeley AI Research, 2019. Caffe webpage. Available from: https://caffe.berkeleyvision.org/ [accessed 11.07.2019]

NumPy, 2019. NumPy Homepage. Available from: https://www.numpy.org/ [accessed 11.07.2019]

Gazi Yalcin O., 2018. Image Classification in 10 Minutes with MNIST Dataset. Available from: https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d [accessed 10.07.2019]