

Training an object detection AI model on a custom dataset



Whitepaper

KEBA[®]
Automation by innovation.

Training an object

Re-training a pre-trained model on a custom dataset is essential in AI vision processing because it enables the model to adapt to the specific characteristics, patterns, and features of the target application. While pre-trained models provide a strong foundation by learning general features from large-scale datasets, fine-tuning them on domain-specific data ensures improved accuracy, robustness, and relevance to the specific tasks at hand.

In this example, we are going to train an AI object detection model to recognize chocolates which are moving on a conveyor belt. Like in the learning process of a young child, intelligent machines also need to learn which objects are present in a picture and how they can be distinguished from each other. This means the AI model's training algorithm is given a large set of images containing the objects it needs to recognize. From these images, it learns to correctly distinguish the objects from the background and from other objects. The model is then created from these recognitions and used in the AI runtime application. For example, for our machine to learn to recognize chocolates, it needs a large number of images of the chocolates taken from different angles, on different backgrounds, exposed to various light conditions and in different positions, colors, and details. Only then can it recognize which characteristics distinguish the chocolates from other objects. The number of images needed for training always depends on the use case and how consistent the environmental conditions such as light or variations in the objects are in the real application. In our case, we expect reasonably stable lighting conditions, so about 500 images of chocolates are enough for sufficiently good results for this intended use case. However, large models with many different objects that are to be used in all environmental variants, e.g. also outdoors etc., are often trained with tens of thousands of images.

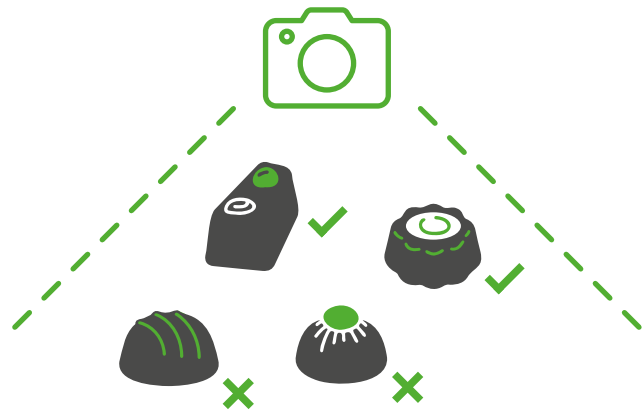
The training process is usually the most time-consuming part in creating an AI application. This is mainly because of the dataset size. Larger datasets generally require more time to train because the model needs to process and learn from more examples. The training strategy also has a large impact. This includes the choice of optimization algorithm, learning rate schedule, regularization techniques and the hardware used for the training. Experimenting with different strategies may be necessary to achieve the best results (<https://encord.com/blog/fine-tuning-models-hyperparameter-optimization/>).

Training in this example uses supervised learning. This method is one of the three most used ones for machine learning tasks. It uses labeled examples of what you are looking for – representing the “right answer” – to compare other pictures against them. The process goes as follows:

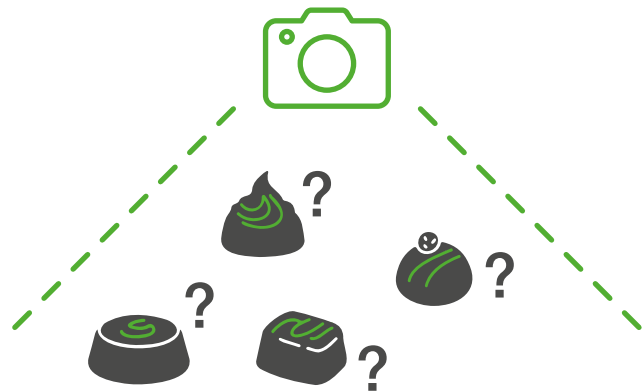
- The machine learning model trains on a set of labeled training data
- It then guesses at the labels of subsequent testing datasets, often in multiple iterations – until...
- it is ready to deploy on the data in the real application

Supervised Learning:

In the first step the labels are given by humans (= training data set)

**Second step:**

The AI gets a real data set and has to decide on its own



As a rough estimate, training a deep neural network on a custom dataset could take anywhere from a few hours to several days for vision processing in an industrial context. In our case, creating the first version of the training set took about eight hours (creating images of the chocolates and labeling them.) and the training of the model by the algorithm itself took about three hours. We had to do three iterations, though, while also improving the first dataset, to get a model good enough for our needs.

Step by step: How to train a model?

-  Create a training dataset
-  Train and test the model
-  Use the model in a real application



1. Create a training dataset

To train a model with supervised learning, you must first create a training dataset. The quality of the model depends mainly on this data. If your data is too sloppy or homogenous, your results could fall apart in the real world. The key is to have the right data and enough of them. The training dataset should contain the information which the model should learn later. In our case we will train an object detection model which will, as the name suggests, detect objects in an image. That means that we need a training dataset with images of the objects we want to detect later. Creating a good dataset is not an easy task. As mentioned above, we used around 500 images for our training. In our use case, the environmental conditions remain quite stable, the background does not change, only the light does. We have captured images under various lighting conditions, such as daylight and artificial lighting. If other objects, like employees' hands or incorrect products, might appear within the camera's view, it's important to include images with these objects in the training dataset as well. However, since we do not highlight these objects, the model learns that the objects are not relevant. This means that false detections can be avoided. False detections can always lead to problems with AI applications. If a model has not learned an object (it was not visible on the images in the training dataset) it will typically try to assign the object to a known object class. To prevent this consistently, objects that may appear in the application should also appear in the training data so that the model can learn that they are not relevant.

The first step in creating a dataset is to capture real world data or generate the training data using CAD models or AI image generation models instead of using real images. In this example, we used real world data. You can do this by either recording a video or taking images. The next step is to pre-process the data. In our case the object detection runtime application works with images with a size of 640x640 pixels. So, we wanted to train the model with images of the same size, but the camera we used for capturing the images didn't support this size. For that reason, we had to crop them into the right format.



The next step is to label the images. Labeling of data in AI applications refers to the process of assigning meaningful and relevant labels or tags to different elements or examples within a dataset. This is a crucial step in supervised machine learning, where the algorithm learns from labeled examples to make predictions on new, unseen data.

Of course, **there are special tools** (<https://aimodels.org/open-source-ai-tools/data-annotation-labeling-tools-machine-learning-datasets/>) that support the user in this task. In our case, we have to mark the object to be detected, i.e. the different types of chocolates, in all the training images. Each chocolate type is assigned to an extra class, so that the model can also recognize them later as extra types/products. This information is saved as a text file. The training algorithm then reads the original image and the text file with the label information and can thus learn the relevant objects in the images.

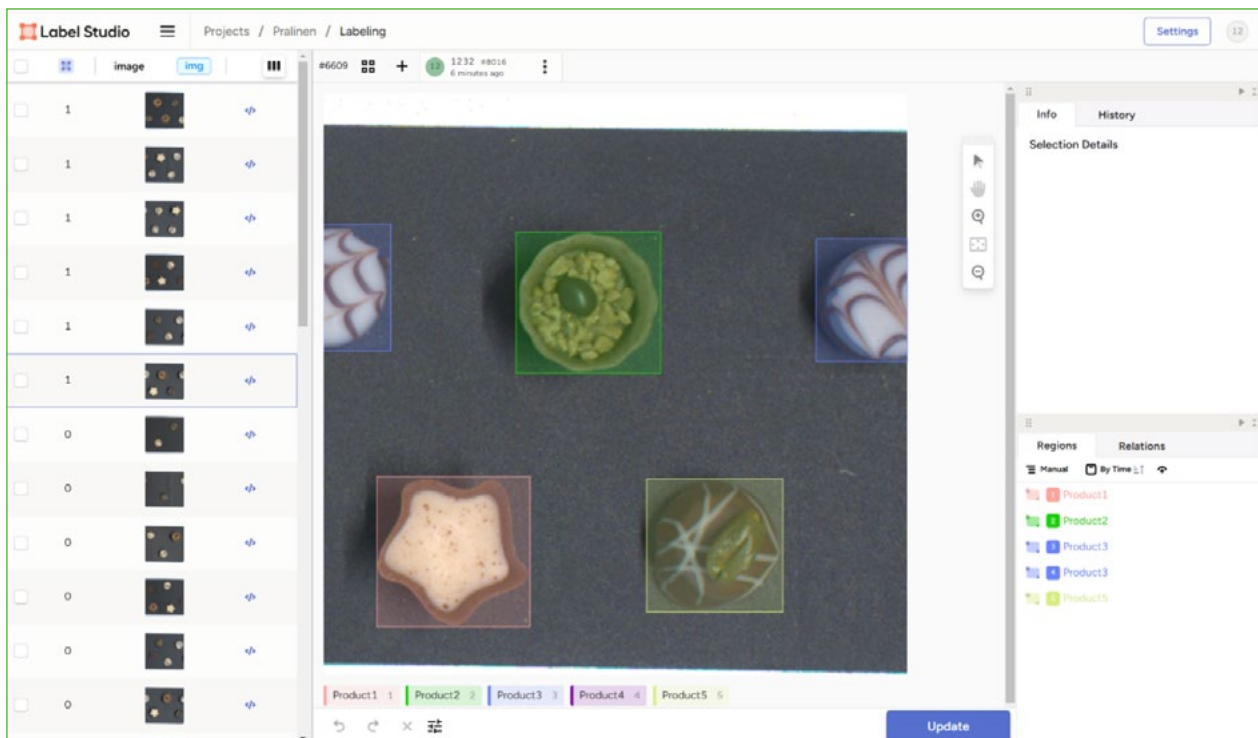


Figure 1: Labeling Images. In the left panel one can select the image to create labels for. The middle panel is where the labels are manually created and the right part overviews the existing labels of the current image.

Before we get to the last step, we can use a trick to generate more training data.

Data augmentation can be used to multiply the already labeled training data. This involves changing the existing images. The changes can be, for example, image rotation, changes in image brightness or color temperature, blurring, noise adding, zooming in or out.

We use a specially created script for this and can thus increase the number of training images by a factor of five using simple image processing techniques. This turns our original 500 training images into 2500. since these operations are applied to the already labeled data, the labels can also be easily adjusted automatically if necessary, saving the effort of labeling these images. Instead of applying a change to each image, we randomly applied augmentation techniques to the images to achieve the desired number of images.

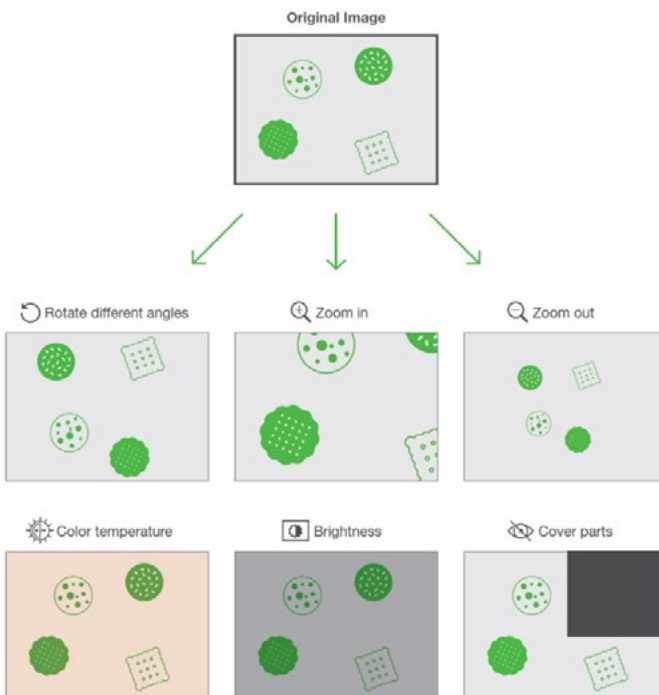


Figure 2: Data Augmentation

Once you have well-curated and prepared data, the last step is to split your dataset into training and validation data. It might require a ton of work to collect and label that much data, but it will be worth it. In rigorous experiments the dataset is typically split into three subsets: training, validation and testing set. The largest set, the training set is used for the main job – training the model.

The training progress is monitored using the validation set. Finally, after the model is trained, the third set is used to get independent measures on how well the model performs. This is important because the validation set is typically used to assess how well the model is trained, determine when training should stop, and tune hyperparameters. However, since it is effectively used as the optimization target, it doesn't offer an independent judgement of the model's quality.

Another important property of a good training data set is its statistical balance. It is important to have the classes equally represented (similar count of samples from each of the classes) such that the model does not pathologically converge to prediction of the most represented class (e.g., in two-class classification, a model that always predicts a class that takes 95% of the dataset has only 5% error).

Any questions?

Simply make an appointment. Our experts are guaranteed to work with you to find the perfect individual solution for your company.

Your Contact Person:

Stefan Fischereeder

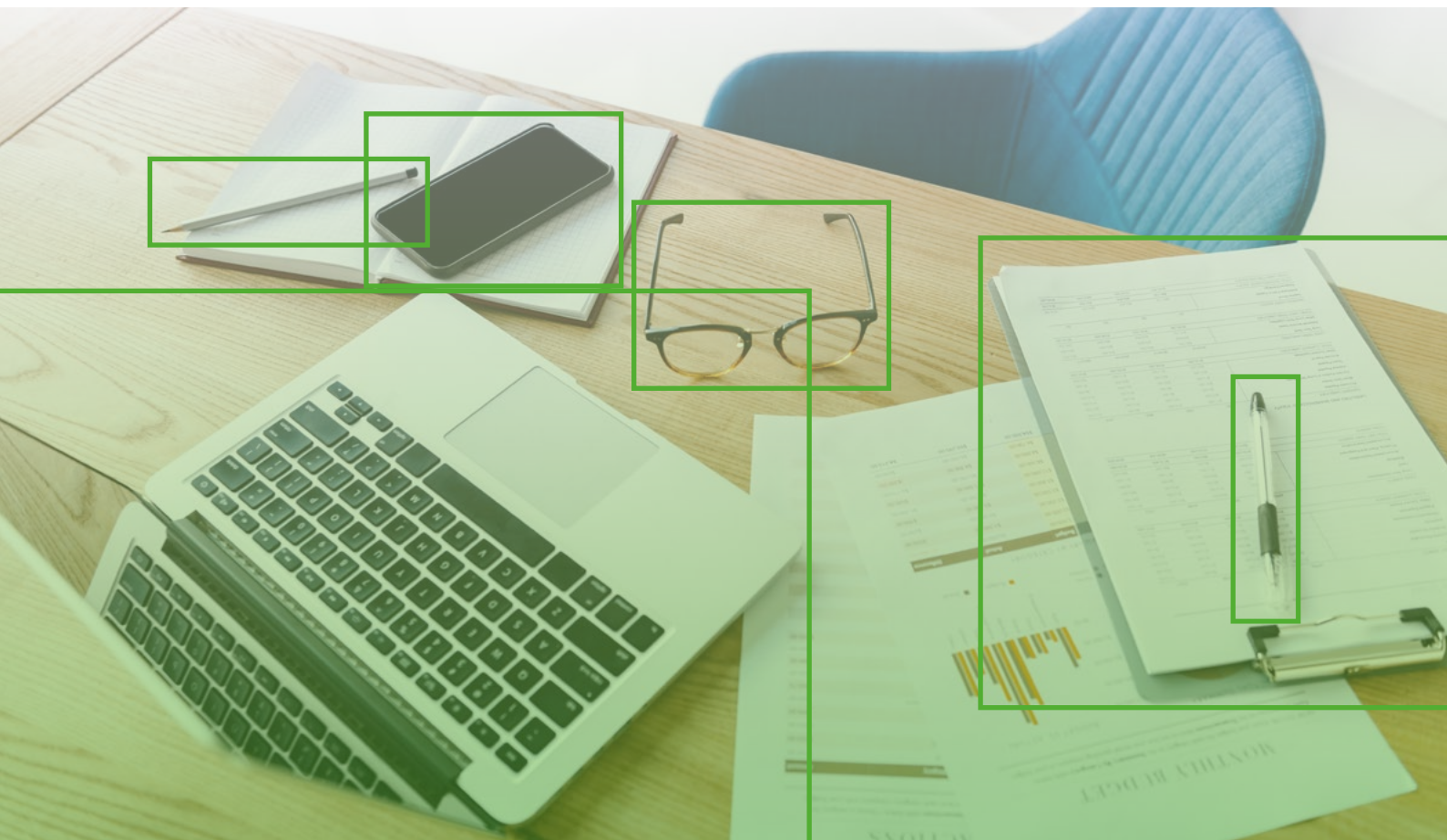
Productmanager Industrial AI

 fsd@keba.com  +43 732 7090 22723

2. Train and test the model

As soon as we have our dataset, we can start the training of the model. For that purpose, a computer with a dedicated GPU is highly recommended, because this will speed up the training a lot. The training can last from several hours to many days. Multiple iterations are essential in training an AI model to ensure it effectively learns from the data, gradually optimizing its parameters. Each epoch allows the model to observe the entire dataset, adjusting its internal parameters through optimization algorithms like stochastic gradient descent. The process aids in generalization, enabling the model to perform well on new, unseen data by avoiding overfitting and underfitting. Complex patterns in the data may require repeated exposures for the model to grasp and incorporate them.

After the training is finished, it is time to test the model against the real data. Images from the real application are used to test the trained model. This is very important in order to obtain valid test results. Just as for training the model, there is also a separate application for testing it. The model to be tested and the test images are then transferred to this application. The application then performs object detection on the test images and saves the images with the recognitions. These images can then be checked automatically by the application or manually by the developer. The assessment of whether a model is good enough for the application usually lies with the developer. For certain applications, it is ok if the recognition does not always work 100%, as possible errors in the application can be easily intercepted. If you can get by with less than perfect recognition results, you can often save yourself a lot of training effort. In our example, errors or false detections must not be allowed to occur, in which case very careful testing is required and more effort may have to be put into training. If you are satisfied with the results, i.e. all objects were recognized with the desired accuracy and there were no false detections, you can proceed to the next step, namely transferring the model to the AI chip. If the test results are not good enough, you must repeat the training process and improve your data set.



3. Use the model in a real application

The last step is to port the GPU trained model to the AI chip. This has to be done because the GPU model is working with floating point values and the AI chip inside the KEBA module can only process integers, (this is not a peculiarity of our module, almost all edge accelerators can only process INT8) so it can achieve a high frames- per-watt performance and thus work in a resource-saving way. Quantization from FP32 to INT8 naturally results in a loss of accuracy. Therefore, after quantization, a calibration of the model with images from the training data set is performed. This step is automatically done by the toolchain provided by the chip manufacturer. After the quantization process is completed, you should test again to check whether the recognition still works with the desired accuracy. If not, you will need to retrain once more, but this is usually not necessary as the loss of accuracy is usually manageable due to the advanced algorithms that now exist for this process. The last step before the model can be used is the compilation for the chip. This step is also done automatically by the toolchain. Now the model is ready to be used on the KEBA AI module.

If you are interested in what is needed for this, we suggest reading another white paper we have written on how to create runtime applications:



Download Whitepaper:

"Create an object detection runtime application"

Conclusions

Training an AI object recognition model has become remarkably simple. Progress extends beyond AI capabilities to include user-friendly tools and streamlined processes. In our case, an apprentice handled the entire application - from data generation and model training to creating the PLC runtime application, while our AI specialists and application engineers provided support. This highlights that specialized know-how is no longer an absolute requirement for training AI or building AI applications. Furthermore, the deep integration of the AI algorithms and models, as well as the vision processing functionalities in the KEBA control system simplify the application creation. The complete AI application can be created directly in the PLC code, which makes it much easier, especially for less experienced developers. AI has a multitude of intriguing use cases for addressing real-world challenges in industrial environments, including those that were previously deemed unsolvable. KEBA has the experience of helping customers solve these problems and offers powerful AI inferencing hardware, software functionalities and support in engineering.

More information

**Product Page:**

AE 550

www.keba.com/ai-extension-module**Interview:**

AE 550

www.keba.com/ai-in-the-industry**News & Blog:**Latest news, press releases
and interesting factswww.keba.com/ia-blog-ai



KEBA Industrial Automation GmbH

Reindlstraße 51, 4040 Linz/Austria, Phone +43 732 7090-0, keba@keba.com

KEBA Group worldwide

Austria / China / Czech Republic / Germany / India / Italy / Japan / Netherlands / Romania /
South Korea / Switzerland / Serbia / Taiwan / Turkey / United Kingdom / USA

www.keba.com



Automation by innovation.