Welcome to the Technical Guide of WhatToLabel

WhatToLabel helps companies build better deep learning models by improving datasets. Our solution can reduce data annotation costs, improve generalization and help to find edge cases.

We currently support images as well as video frames as input data. LiDAR support is in development. Audio and Text support will follow in late 2020.

This technical documentation shall explain in more detail how our solution works.

General Information

From our experience, there is no such thing as one solution to fit all demands. Therefore, we developed our software with three goals in mind:

Extendibility

- We regularly explore and add new features such as embeddings or filters based on customer request and our research

Performance

- We optimized the algorithms to use less compute and memory resources and work with millions of samples on a single machine

Transparency

- We strive to provide as much information about the analyzed dataset and the filter process as possible to our customers

Recommended Hardware

We test the filtering solution in our Google Cloud environment on a system with the following specification:

- 8-cores, AVX2 support
- 16GB Ram
- P100 or V100 GPU

We mark features which are currently in development and not released in our production docker container with ***Experimental***

Related Work

The WhatToLabel team keeps up with new publications in the area of active learning, self-supervised learning and data mining to provide the best results for our customers. In the following, we show a selected list of interesting publications related to the field. We are in constant exchange with industry-leading research labs to improve our algorithms and implementations.

Self-Supervised learning

Contrastive Multiview Coding, 2019

Active-learning

Discriminative Active Learning, 2019

Variational Adversarial Active Learning, 2019

Dataset redundancy

The following papers from Google and Nvidia validate our results that removing data redundancies can improve test accuracy.

Semantic Redundancies in Image-Classification Datasets: The 10% You Don't Need, 2019

Training Data Distribution Search with Ensemble Active Learning, 2019

> Note: we are not always able to reproduce the results from the publications. Our production docker container only contains filters that run through our internal benchmarking system.

The WhatToLabel Solution

Compared to existing solutions we stand out:

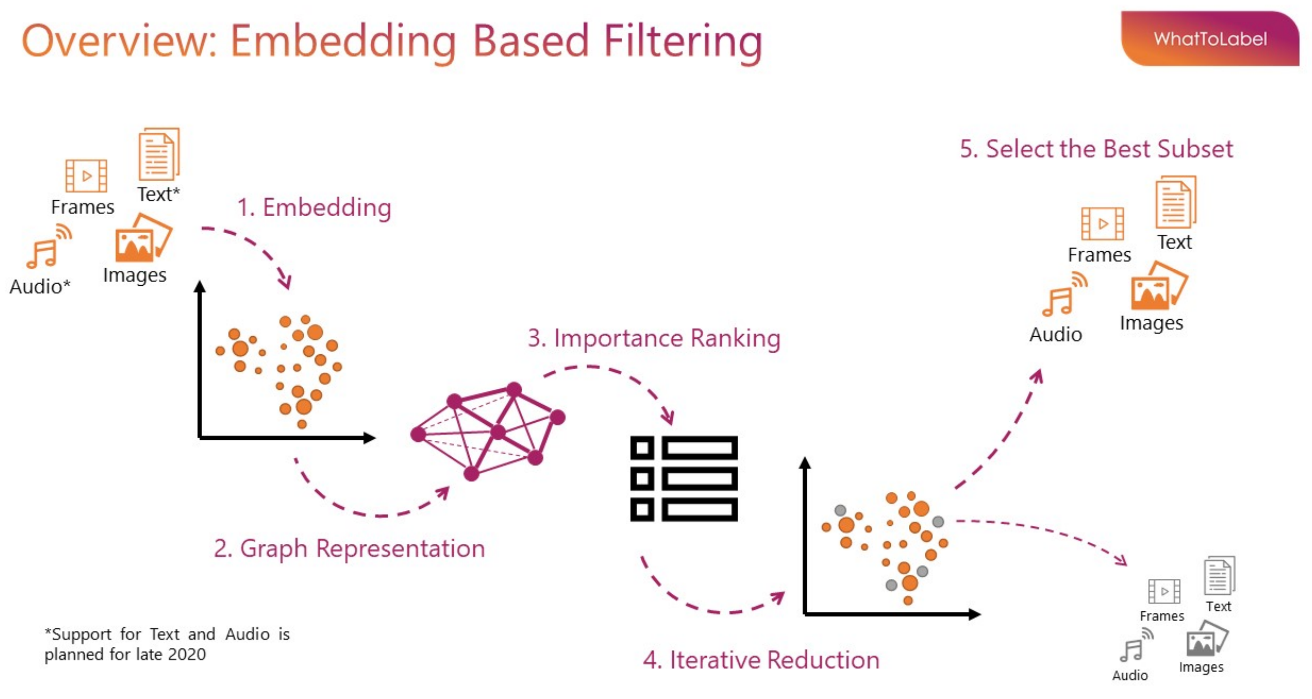
- Our **professional** work and **industrial-grade support**
- We **ensure reproducibility** and provide **transparency** of our filtering solution
- The **unsupervised** nature of our filters make it very **easy to integrate** into existing pipelines

High-level Overview

Our dataset filtering approaches can be divided into two main groups. *Embedding based filtering* and *Learned filters*. The first group consists of two steps, first getting a good embedding and then selection of important samples. The second group is based around filters that learn to separate important from less important samples. Note, that all our filters work on a semantic level.

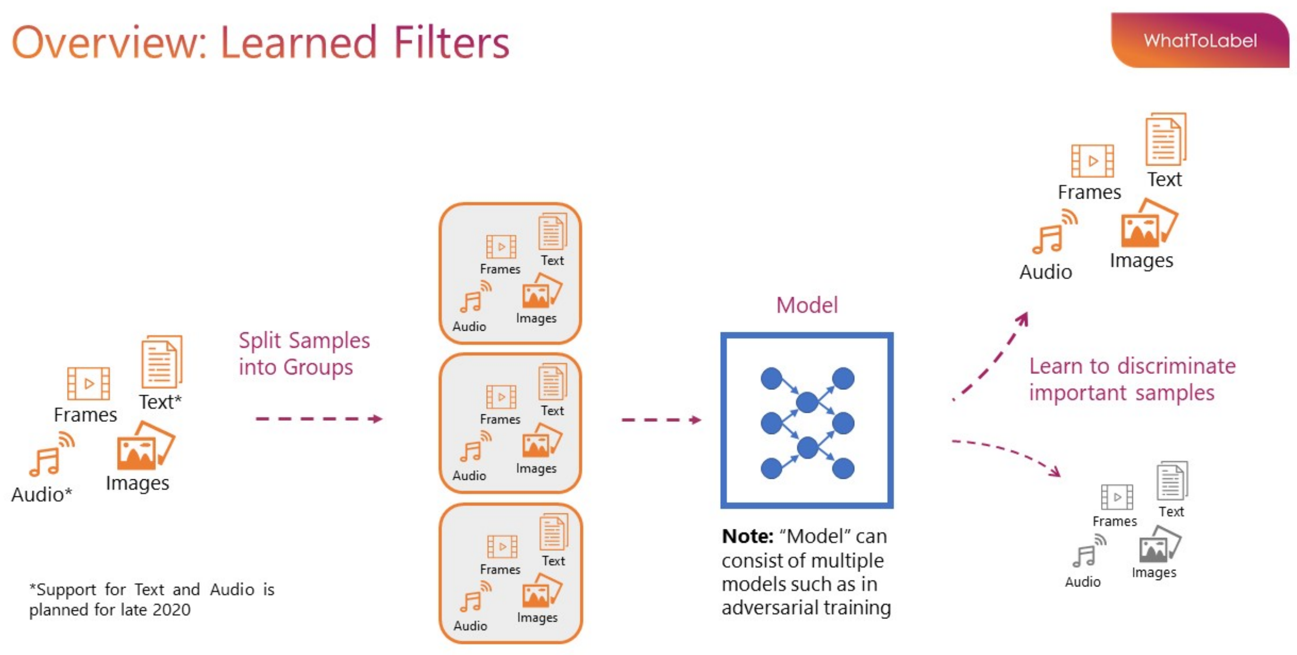
Embedding based filtering

- Filters using either pre-trained or fine-tuned embeddings
- This group of filters works for small datasets and is usually very fast.



Learned filters

- Filters which find the most relevant samples by learning the distribution of the dataset
- The learning process takes more time and requires more samples.


Overview: Learned Filters
WhatToLabel
Frames
Text*
Audio*
Images
*Support for Text and Audio is planned for late 2020
Split Samples into Groups
Frames
Text
Audio
Images
Model
Note: "Model" can consist of multiple models such as in adversarial training
Learn to discriminate important samples
Frames
Text
Audio
Images

Methods

In the following section, we describe the different filtering methods in more detail.

Embedding based filtering

Embedding

Our framework provides different embedding algorithms. For images, they typically match common architectures such as resnet34 or resnet50. We additionally optimized our embeddings for a **high accuracy** and **low dimensionality**.

Low dimensionality has multiple advantages. On one hand, it reduces memory usage and calculation time of subsequent algorithms. In high dimensions, distances become less meaningful. This problem also known under terms such as the curse of dimensionality can be reduced by using low dimensional embeddings. Especially, when removing 90% or more of the initial dataset the low dimensionality of the embedding is crucial.

> E.g. our pretrained resnet50 like embedding has an output dimension of 64. Accuracy on ImageNet (val): Acc@1 77.368 Acc@5 94.054. We achieve this by extensive pre-training on various large-scale datasets.

Model	Top1 Accuracy	Top5 Accuracy	Output Dimension
ResNet-50*	76.15	92.87	512
ResNet-50 (ours)	77.368	94.054	64
ResNet-101*	77.37	93.56	2048

*Numbers taken from https://pytorch.org/docs/stable/torchvision/models.html

Code used for ImageNet validation accuracy:

Evaluated using ILSVRC2012 validation set and code from: PyTorch Github

Experimental: We have pre-trained embeddings which can be fine-tuned using self-supervised learning on large enough datasets (currently using Contrastive Multiview Coding, 2019). Self-supervision can be used if no labels are present. In case there are (weak) labels, they can be used instead. Fine-tuning the embedding can improve filtering.

Distances and Graph representation

Depending on the algorithm for filtering a graph will be created. E.g. *Adjacency* based filtering removes the shortest edge of a graph iteratively.

For smaller datasets a fully-connected graph and larger datasets, a local neighborhood graph will be created. The latter one is using Dijkstra to find the shortest paths between nodes. This needs additional compute and memory.

For the distance metric either L2 or normalized L2 can be used. In case of normalization the software scales each sample to unit L2 norm.

Filtering Algorithms

Adjacency

Removes the shortest edge of a graph iteratively. This approach works best if the final dataset is above 50% of the initial dataset size. Removing semantic similar samples can remove overfitting and improve accuracy.

Divide and Conquer

This filter can be combined with any other filter. First, we use density-based clustering such as spectral clustering on the embeddings. Then, we apply the filter on the individual clusters.

Coreset

Based on greedy coreset algorithm from *Discriminative Active Learning, 2019*. Our implementation uses C++ and OpenMP. The coreset filtering approach is very fast. Combined with our low-dimensional embedding it also provides good results when removing 90% of the samples.

Learned Filters

Our background with generative adversarial networks (GANs) and the respective adversarial training helps us explore new filtering approaches. Learned filters a very interesting since they work with unlabeled data in specific domains where pre-training an embedding is not feasible.

VAE-GAN (*Experimental*)

Based on *Variational Adversarial Active Learning, 2019*. Trains variational auto-encoder with discriminator to learn data distribution.

> Filter is in the experimental state since results from paper not yet reproducible. We're in discussion with the author to sort out the issues.

Filtering Results

Filtered Dataset

We provide a final list of *important* samples. This list is provided in a `.txt` file with each line consisting of a filename. The filter software additionally allows for automated copying of the filtered dataset to the output folder.

Dataset Insights

Filtering a dataset can have a strong effect on the subsequent processes such as training a deep learning model.

To further understand what exactly happened with the dataset and provide general information about the dataset we share dataset insights with our customers. Our goal is to provide meaningful information about the collected raw dataset before annotation or training processes.

These insights include plots of the embedding. We currently use UMAP dimensionality reduction which is commonly used for larger datasets.

Additionally, we provide examples of kept samples as well as nearest removed neighbors to understand why certain samples have been removed.

Histograms of distances can provide insights on the diversity of the filtered datasets (before and after filtering).

Further Questions?

Don't hesitate to reach out via e-mail: igor@whattolabel.com