FALLBACK CHAIN CONCEPTS

The concept of fallback chains can be used as an incubator for continuously improving models over match cases while at the same time providing maximum fault tolerance when these models fail. The idea is simple: don't put all eggs in the same basket. Even when a particular match configuration generalizes well over wide range of match cases, it can still fail at a new case it wasn't trained on. A chain can help leverage this with a second simpler configuration while at the same time collecting data to improve the first. But let's get into some details to understand all of this better.

At the very minimum and simplest matching possible, we should have a match configuration file here called **config** and a target data like image, text, cascade or network here called **data**. The two combined together constitute a single **model** which in the simplest case we consider should be applied to a single match case, i.e. a single use case when we need to find a needle in a haystack. Using this simplest approach, we will need a new model for each match case we encounter and the number of models can grow a lot with the complexity of any given scenario. It is therefore much better practice (and much more interesting) to attempt to generalize some of the models and apply them to multiple match cases. For instance, we might want to use a single model to match an OK button in various circumstances like different software versions, GUI style, and screen variations.

The available computer vision backends generalize with various degree of success depending on multitude of factors that are not important for this discussion. What is important though is that all of them could eventually fail when encountering a match case they weren't **trained** for, i.e. a case their parameters in their respecive configs are not adapted for. This is fine as long as the models are retrained to include the new sample which might or might not be possible depending on the AI quality of the model and the available data from the previous match cases. Even if the model is adaptable over a large set of match cases, we need to guarantee training it for the new case will not deteriorate its accuracy in the previously collected cases. One way to do this would be to keep a stash of images for each match case the same model should apply to. The obvious drawback of this is that we need to store this separately from the models as datasets to use only on retraining and the increase in the number of match cases can make this very hard to manage. This is where fallback chains come to use.

The idea of a fallback chain is to combine both inferior and superior models in a **chain** starting with the most and ending with the least generalizable ones. A failure in the prediction of a more applicable model will result in a fallback to a less applicable one and the chain of multiple such will only fail if all the models along it fail. The order in the models makes sure that a performance punishment is endured only when the most probable to succeed models don't manage thus trading robustness for performance as efficiently as possible. The overhead in adding a new failed case is also minimized because the minimum requirement would be to add a perfect match method at the end of the chain which is equivalent to one extra line of configuration and one extra image. If more resources are available, one could then **shorten** the chain by retraining the top model, squashing simpler models into more robust ones, etc. The chain also becomes a storage point for collected data or a **dataset** that can be used for future improvement of its top models, be it tracing, training, or testing.

Better models generalize over more match cases and lead to the existence of fewer models in general. However using chains of models makes sure that the matching always performs optimally despite limited resources for such improvement. It allows us to create better and continuously improvable models but to avoid any risk of breakage in all match cases we already apply them to.

FALLBACK CHAIN API

In tradition of not becoming overly verbose, let's quickly glance at interface to use for fallback chains. The entire chain definition is located in a single **steps** file e.g. *chain1.steps* following the philosophy of maximally separating configuration from execution logic. And what is a configuration if not simply an input to a program that has grown too much for simpler means? But philosophy aside, the steps file contains one line for each model to be used as follows:

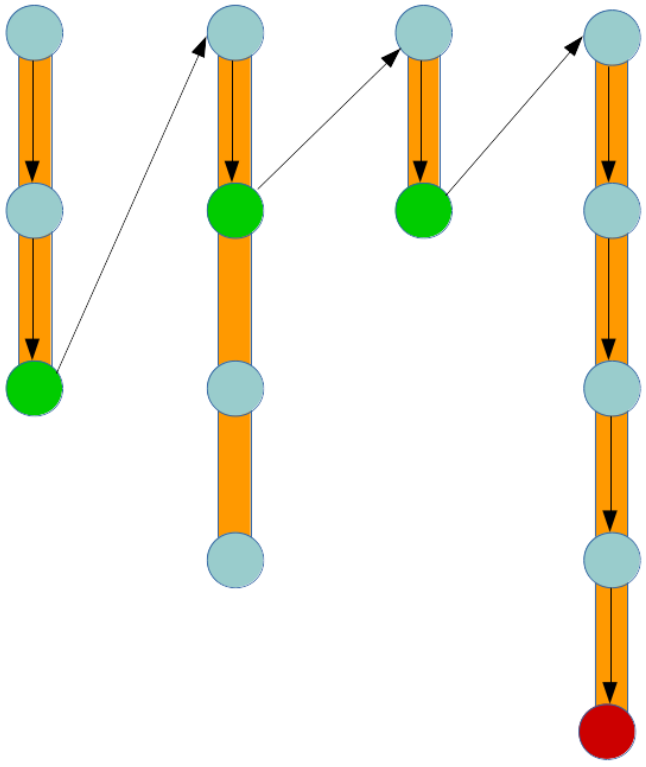
```
<data-file> <match-file>
```

For instance, a chain of two models, one template- and one feature-based would be

```
circle.png template-for-circle.match
circle.png feature-for-circle.match
```

Notice that we use the same data file but nothing stops us from using different ones for each model. We can change configurations, data, or both and add as many lines as we like but it is advisable that we follower the order of most to least applicable models recommended above.

The matcher to use with chains is the **HybridMatcher** and the target type is nonsurprisingly **Chain** which should be initiated with the steps file containing the chain definition. More information about this could be found in the API documentation.



The principle behind the way it works is also immediate – the system will try to match with a model and if it fails it will move to the next available model along the chain. If successful, it will move to the match case and its corresponding fallback chain. If it reaches the end of a chain and has no more models to try, it will fail in the typical way returning zero matches. Whether this is an actual **FindError** or can be tolerated is up to the caller to decide and is thus a subject of external forces that don't concern us here any longer.

Using optimally different models or data for training are some among the many possible techniques to improve the accuracy along a chain. But all of them boil down to machine learning again so if you have the time and want to dive, here is a waypoint. Just some fun thoughts to leave you with.