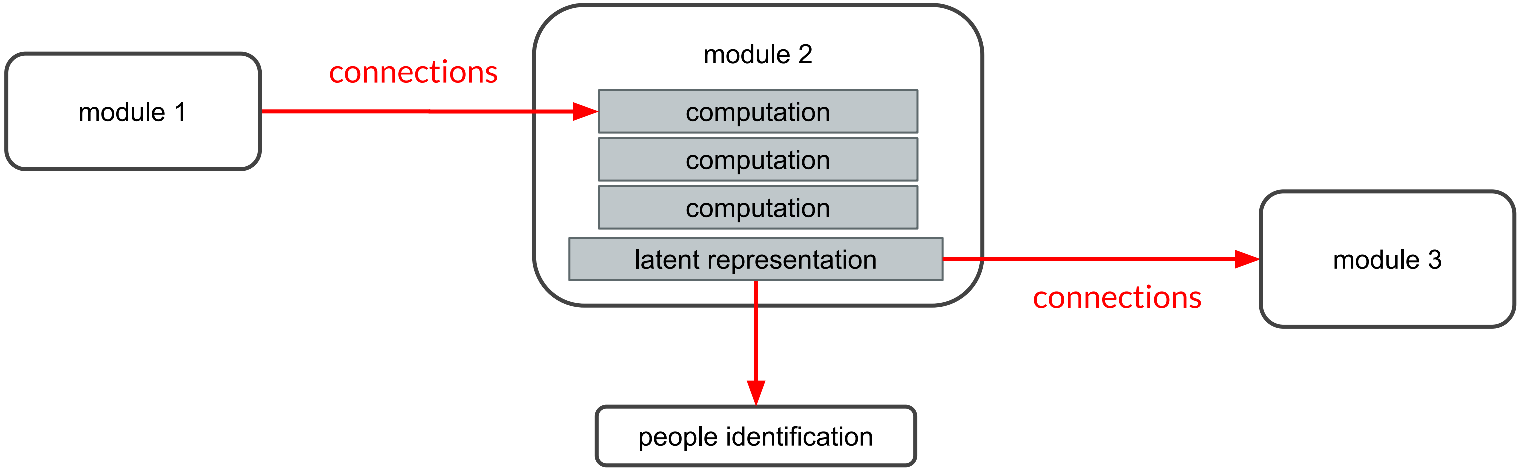
Meaning-Preserving Continual Learning v1

Intuition Pumps

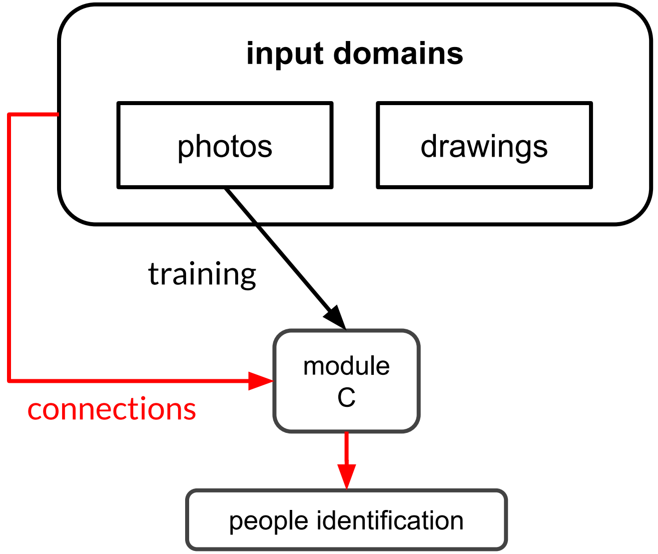
Intuition Pumps (1)

In MPCL, modules compute latent representations, which in turn are connected to:

1. An anchoring function, e.g. people identification ;
2. Other modules.



Intuition Pumps (2)



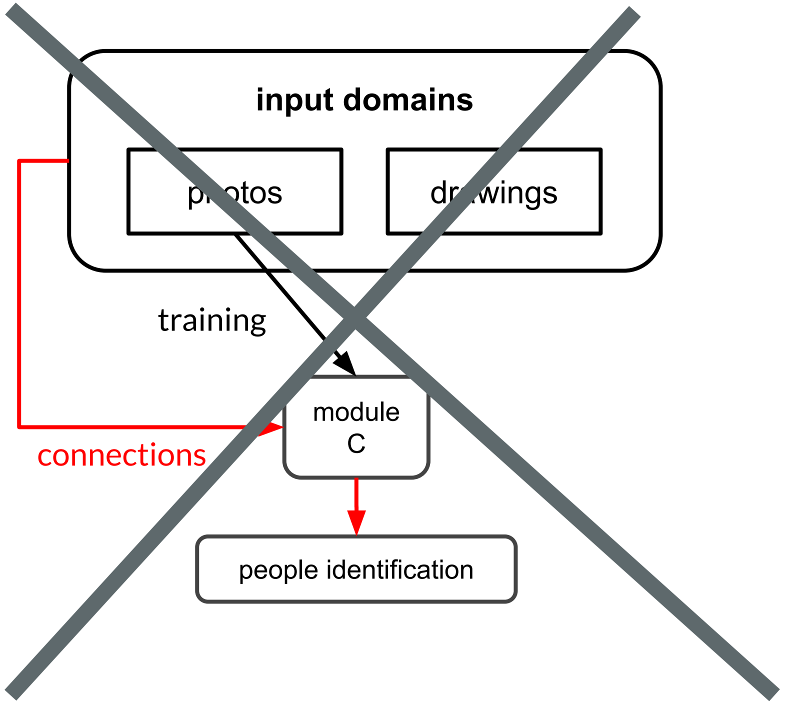
Module C's function is to identify people.

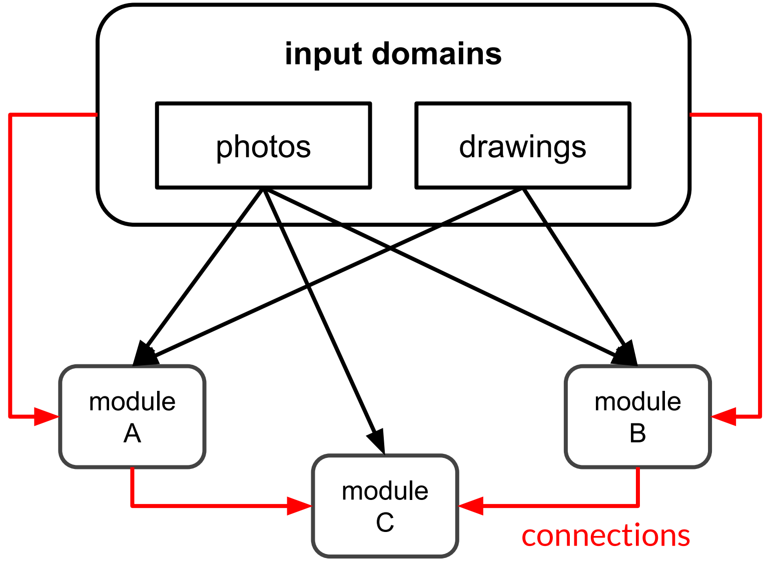
Truism: if module C is trained from photographs only, it is not going to transfer well to drawings.

By expecting module C to generalize to drawings, you tacitly expect C to know what "people" means.

Module C doesn't know what "people" means.
It's not a concept that a basic model can grasp by sitting in a server room.

Intuition Pumps (3)

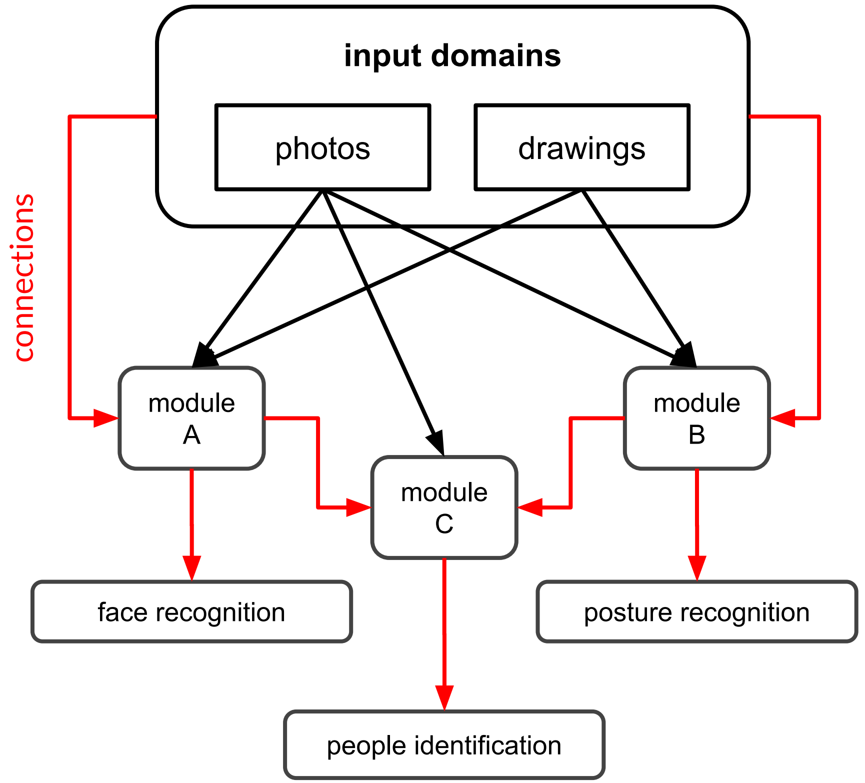




Here too, module C is trained from photographs only.

However, as module A and B were trained from both photos and drawings, module C is a lot more likely to successfully transfer to drawings.

Intuition Pumps (4)



Furthermore, as module A is trained separately via an anchoring face recognition function, and provided it is accurate on both photos and drawings, faces will be represented with the same vectors in photos and drawings (*).
It follows that if module C utilizes John's face and posture to identify him in photos, it will also identify him in drawings.

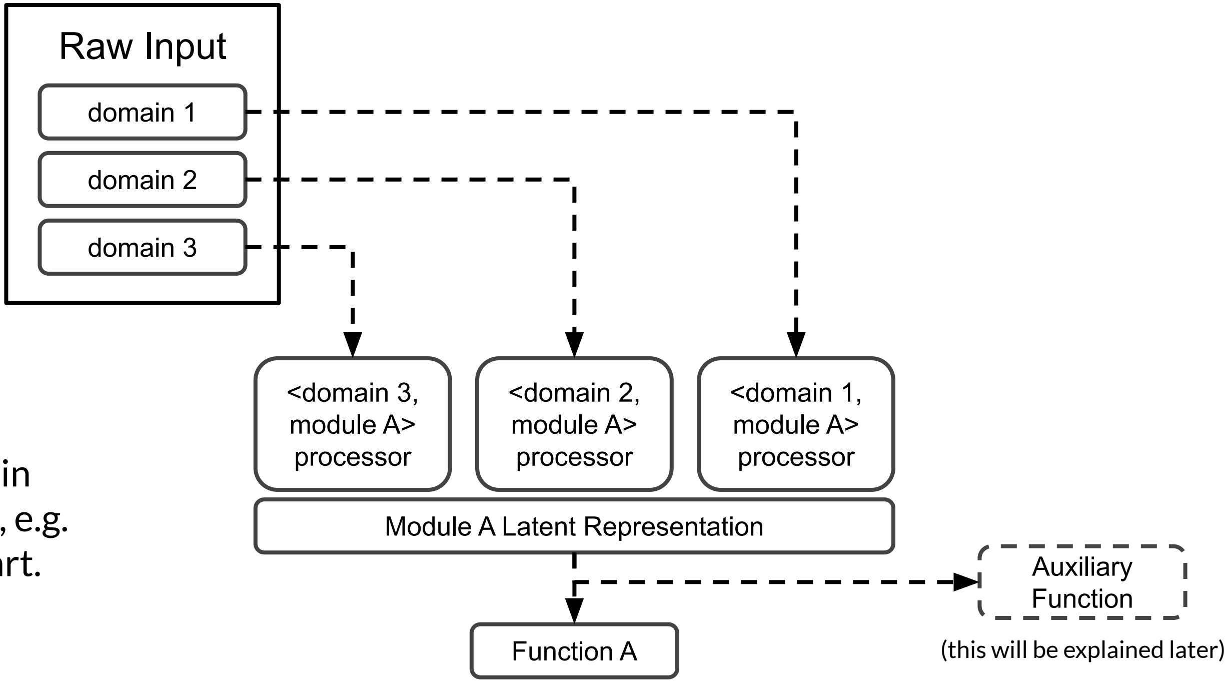
This can be seen as a form of curriculum learning or machine teaching.

(*) under some conditions that are detailed in the LaTex document.

Modules

Module Anatomy

domains are differently distributed

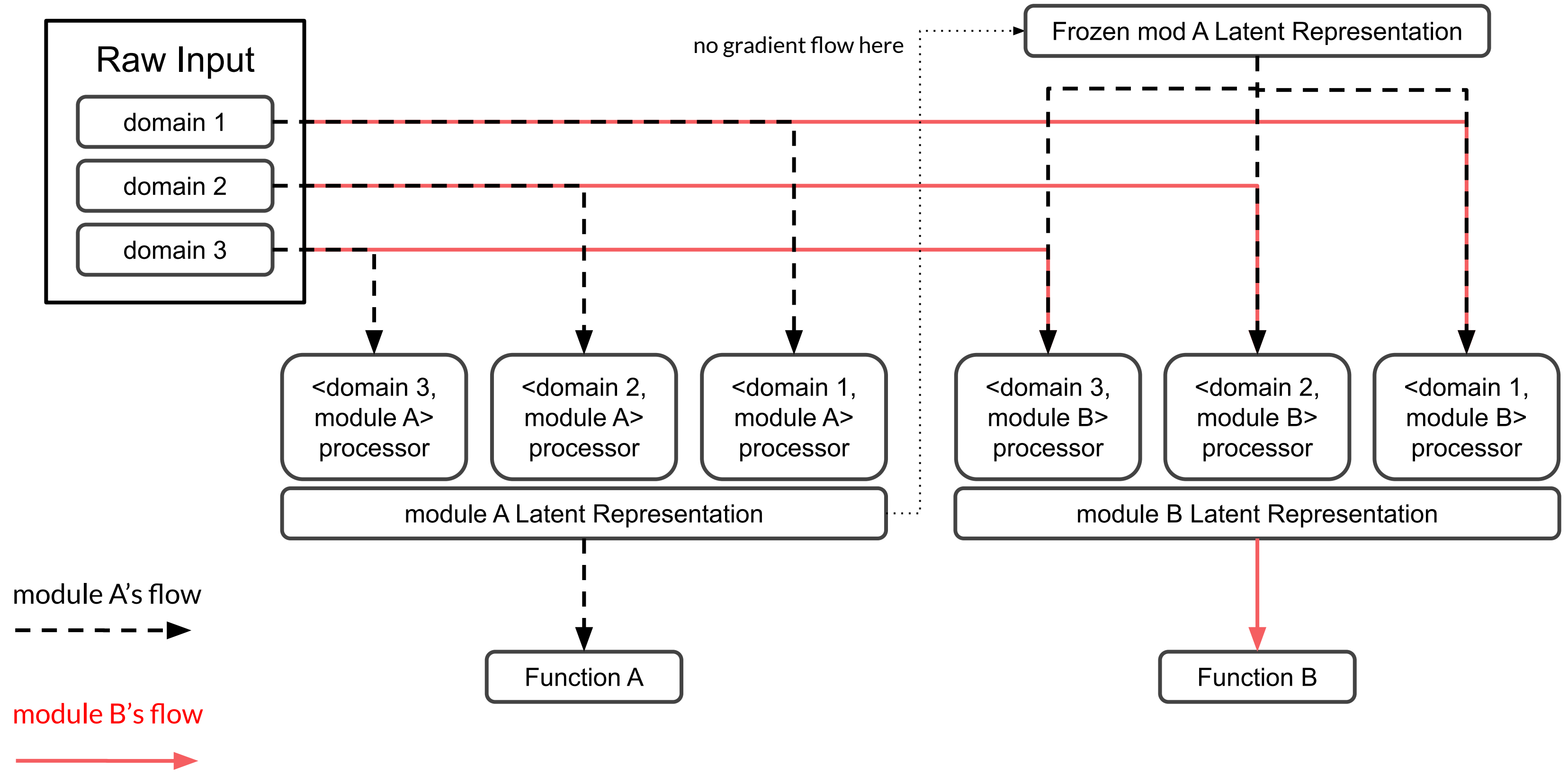

Raw Input
domain 1
domain 2
domain 3
<domain 3, module A> processor
<domain 2, module A> processor
<domain 1, module A> processor
Module A Latent Representation
Auxiliary Function
Function A
(this will be explained later)

Function A is expressed in subjective human terms, e.g. telling cats and dogs apart.

For *HumanMeaning(ModuleA) = Meaning(Module A's Latent Representation)* to hold true, module A's output classes must be accurately predicted* across many domains/contexts.
If there aren't enough domains, there is no guarantee that *Meaning(Module A's Latent Representation)* aligns with *HumanMeaning(moduleA)*.

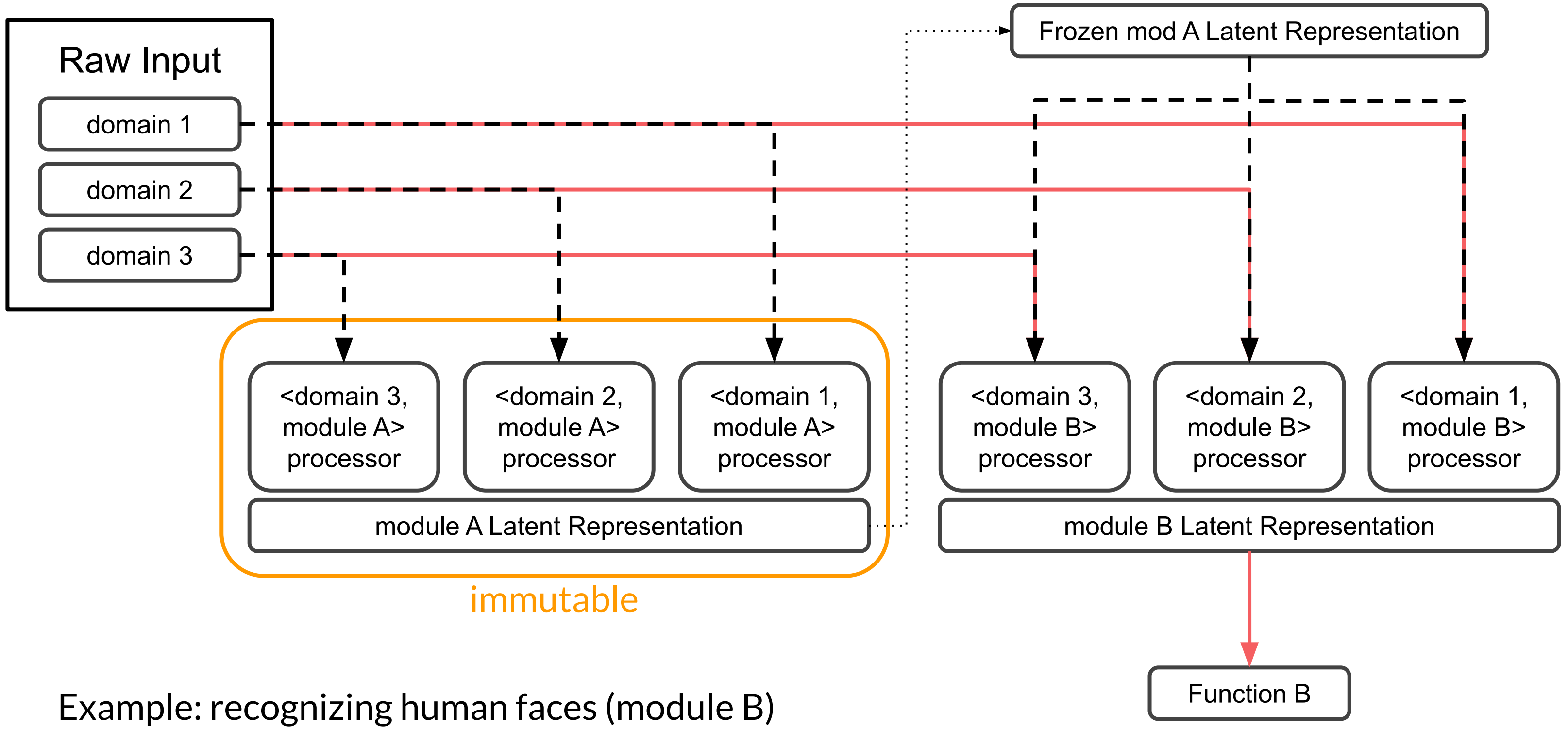
*prediction of classes or numerical values, or rewards from motor goals.

Two-module Scenario


Raw Input
domain 1
domain 2
domain 3
no gradient flow here
Frozen mod A Latent Representation
<domain 3, module A> processor
<domain 2, module A> processor
<domain 1, module A> processor
<domain 3, module B> processor
<domain 2, module B> processor
<domain 1, module B> processor
module A Latent Representation
module B Latent Representation
Function A
Function B
module A's flow
module B's flow

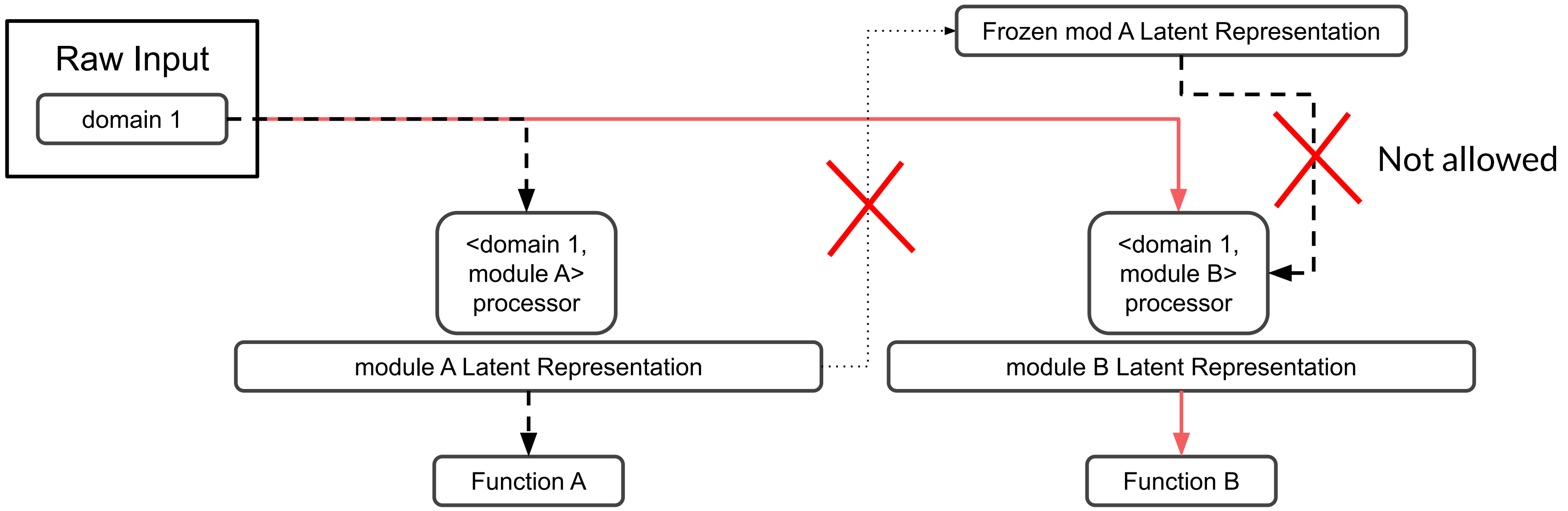
Rules

Rule 1: module A is frozen when training module B


Raw Input
domain 1
domain 2
domain 3
Frozen mod A Latent Representation
<domain 3, module A> processor
<domain 2, module A> processor
<domain 1, module A> processor
module A Latent Representation
immutable
<domain 3, module B> processor
<domain 2, module B> processor
<domain 1, module B> processor
module B Latent Representation
Function B

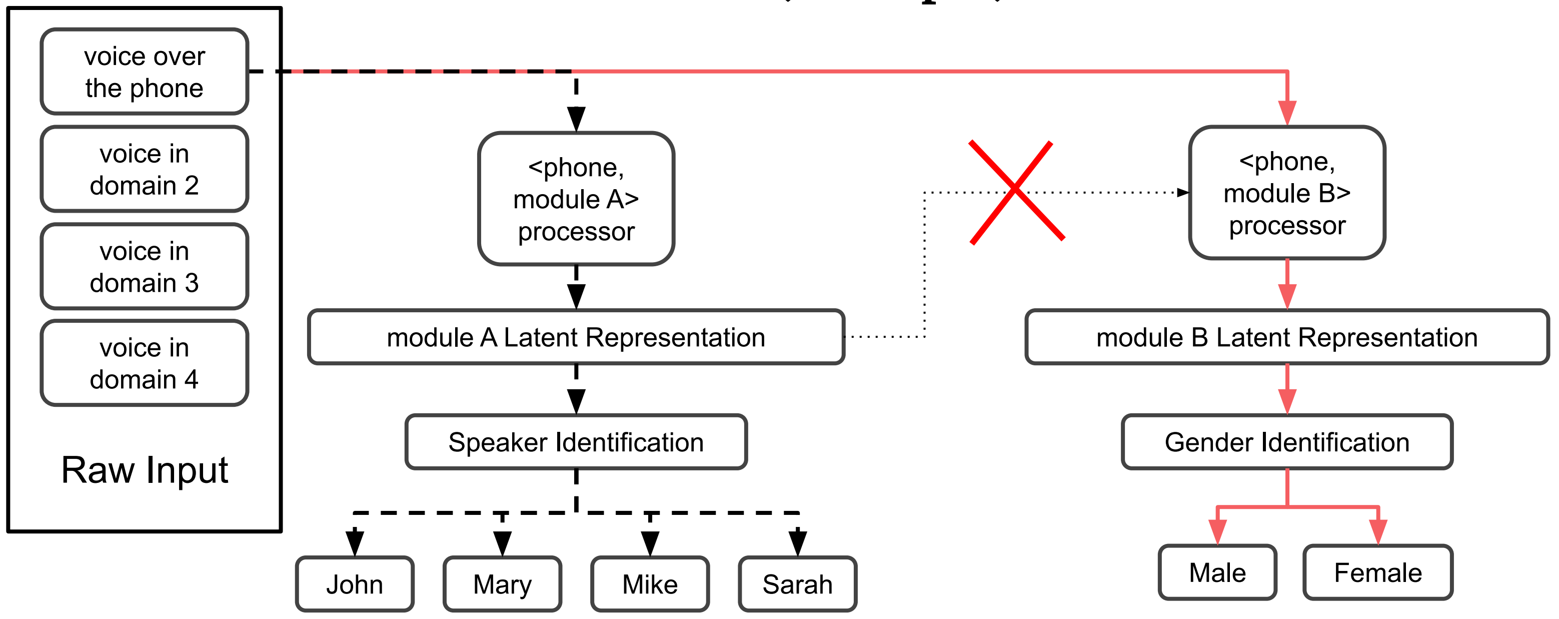
Example: recognizing human faces (module B) cannot interfere with the function of telling cats and dogs apart (function A).

Rule 2: module B cannot utilize module A if training domains were scarce


Raw Input
domain 1
Frozen mod A Latent Representation
Not allowed
<domain 1, module A> processor
<domain 1, module B> processor
module A Latent Representation
module B Latent Representation
Function A
Function B

If there is a misalignment due to the lack of training domains, i.e. *HumanMeaning(moduleA) != Meaning(module A's Latent Representation)*, then the system might exploit a correlation between module A and module B that doesn't exist in our reality. This will create problems when new domains are observed.

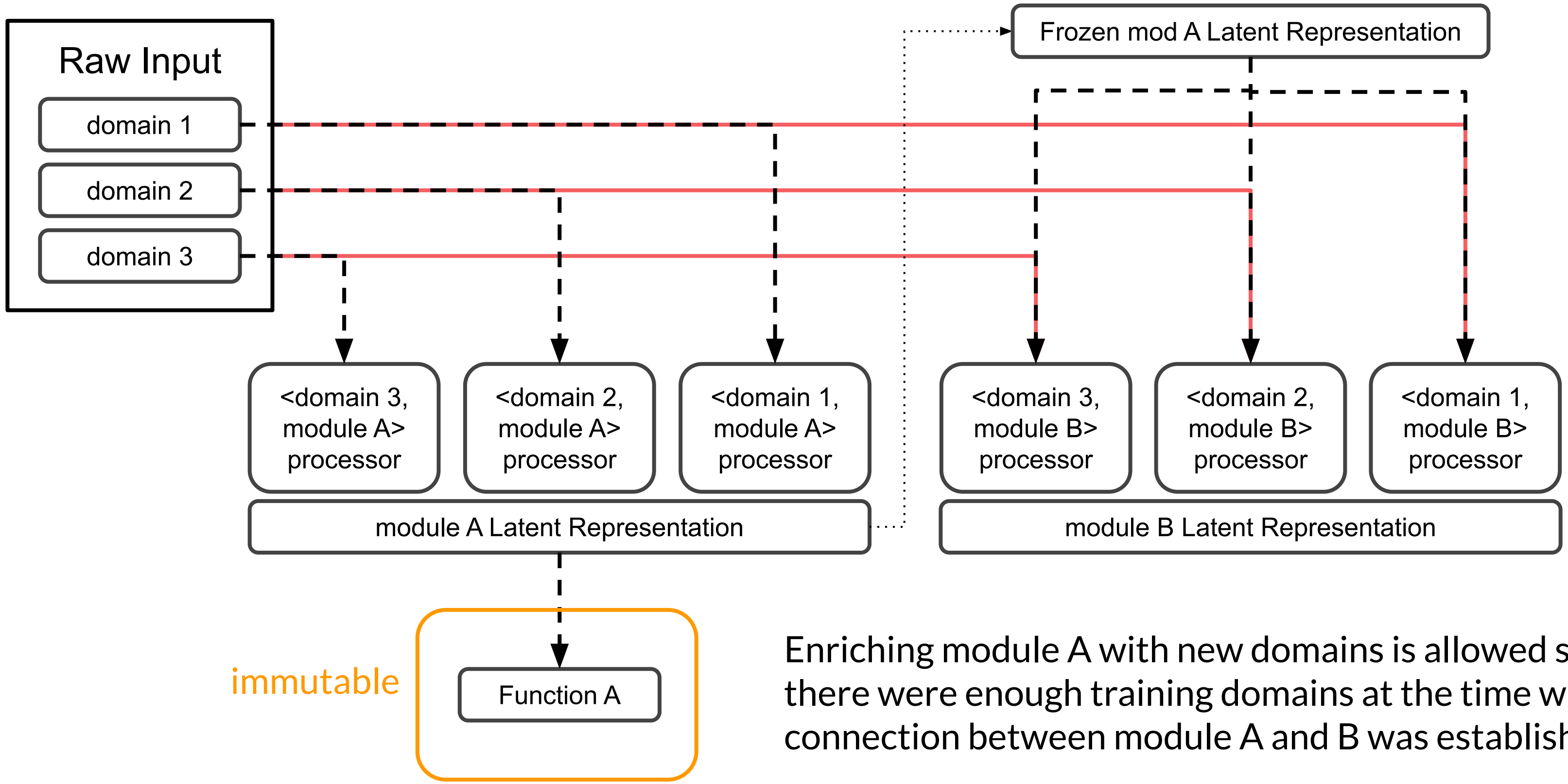
Rule 2 (example)



In this example, module A is trained from phone calls. While we want module A to pick up on vocal characteristics, it might instead pick up on microphone characteristics. If module B utilizes A's latent representation, B might still be able to guess the speaker's gender from phone calls, but it will certainly not readily work in other domains/environments.

In contrast, if A were to be accurate over a variety of domains, its latent representation would contain relevant information about people's identity (according to *our* perspective that *we* provide through labels), and B would make inferences that would readily transfer to domains that haven't yet been encountered, even by A. [accuracy measurement: see slides about domain generalization]

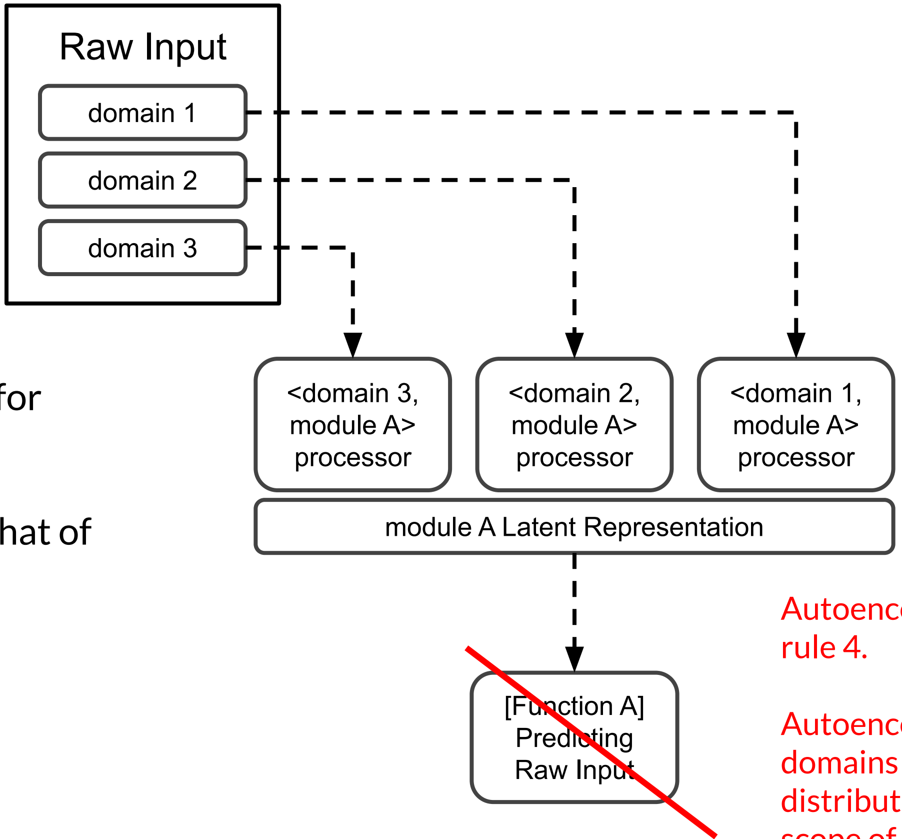
Rule 3: module A is allowed to interfere with module B if it doesn't break Rule 2



Raw Input
domain 1
domain 2
domain 3
Frozen mod A Latent Representation
<domain 3, module A> processor
<domain 2, module A> processor
<domain 1, module A> processor
<domain 3, module B> processor
<domain 2, module B> processor
<domain 1, module B> processor
module A Latent Representation
module B Latent Representation
immutable
Function A

Enriching module A with new domains is allowed so long as there were enough training domains at the time when the connection between module A and B was established.

Rule 4: modules must interpret/abstract their input

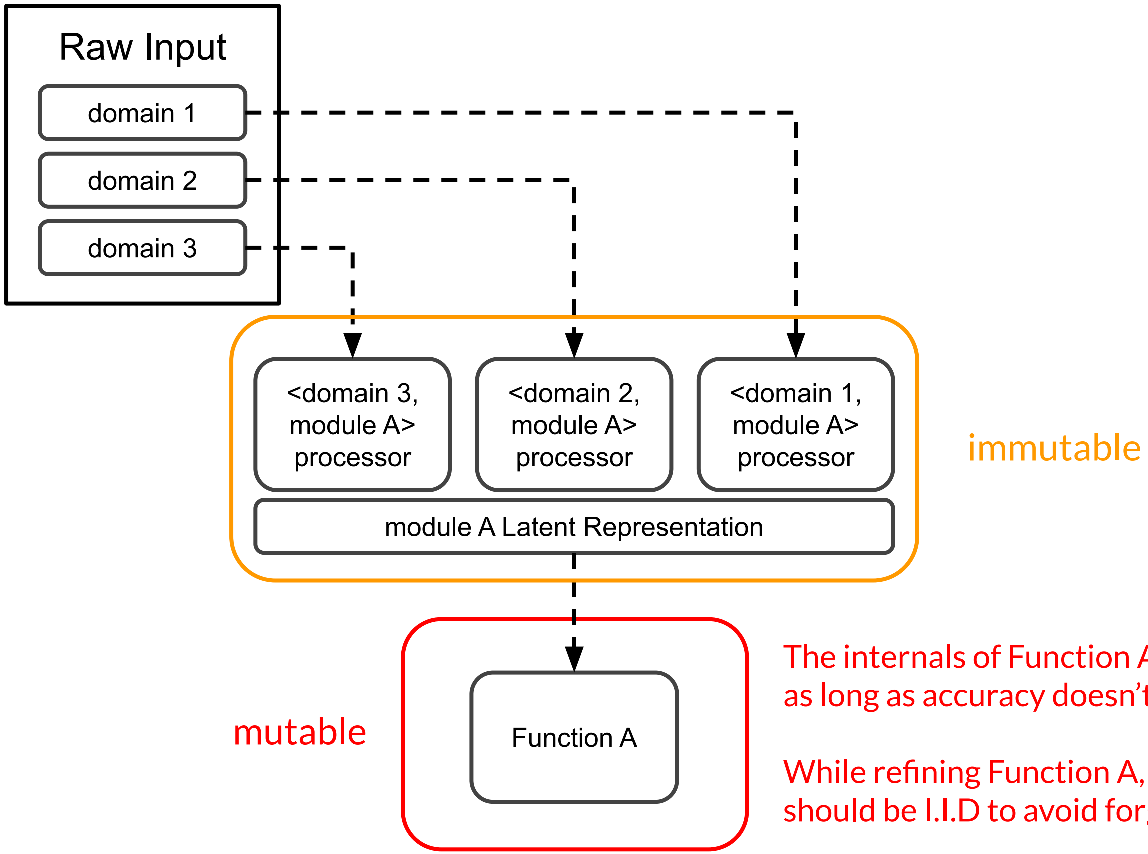

Raw Input
domain 1
domain 2
domain 3
<domain 3, module A> processor
<domain 2, module A> processor
<domain 1, module A> processor
module A Latent Representation
[Function A] Predicting Raw Input

Rule 4 is necessary for domain 1's latent representations to be congruent with that of domain 2 and 3.

Autoencoding is incompatible with rule 4.

Autoencoding might work if domains are identically distributed, but that's beyond the scope of MPCL.

Rule 5: functions' internals can be improved when representations are fixed


Raw Input
domain 1
domain 2
domain 3
<domain 3, module A> processor
<domain 2, module A> processor
<domain 1, module A> processor
module A Latent Representation
immutable
mutable
Function A

The internals of Function A can be altered as long as accuracy doesn't decrease.

While refining Function A, goals/targets should be I.I.D to avoid forgetting.

Rule 6: if a module cannot accommodate new domains, it should be removed

- If a module is accurate on its original training domains, but cannot reach high accuracy on newly observed domains, it means that the module's latent units are not suited for the problem at hand.

- In other words, the learned representations do not abstract away input domains in the same way as we do. The module picked up on different regularities. Due to this misalignment, the module cannot be safely attached with other modules.

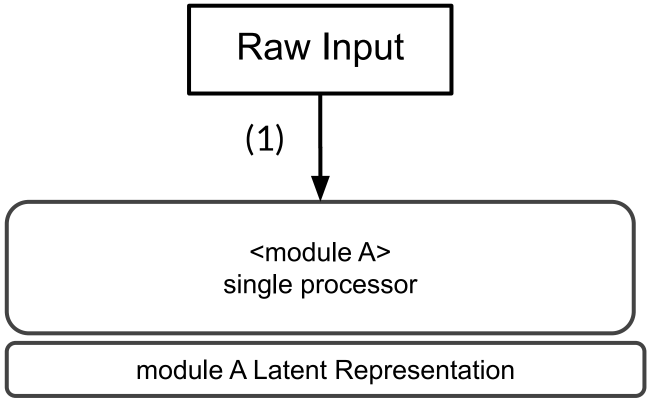
- Counter-measure proposal: train multiple modules for the same task. Pick the module that comes up with the best abstraction hypothesis.

- Measuring a meaningful accuracy is not trivial. (See the slides about domain generalization)

Domain boundaries
and
proto-MPCL

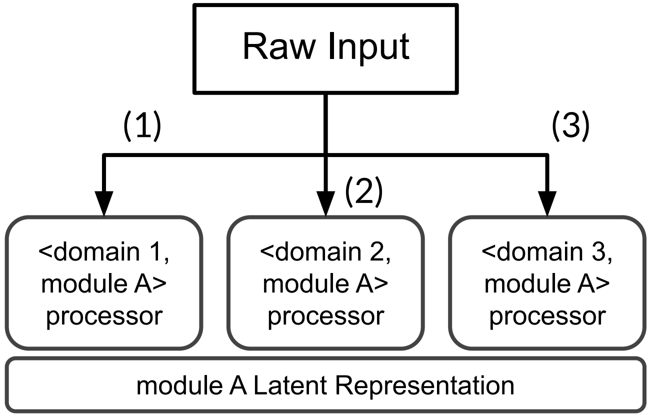
Inside Modules: The Routing Problem

Conventional Continual Learning



It is common for CL algorithms to train a single processor for all domains.
There is no issue detecting domain boundaries and mutualising knowledge, but there is a risk of catastrophic forgetting.

MPCL

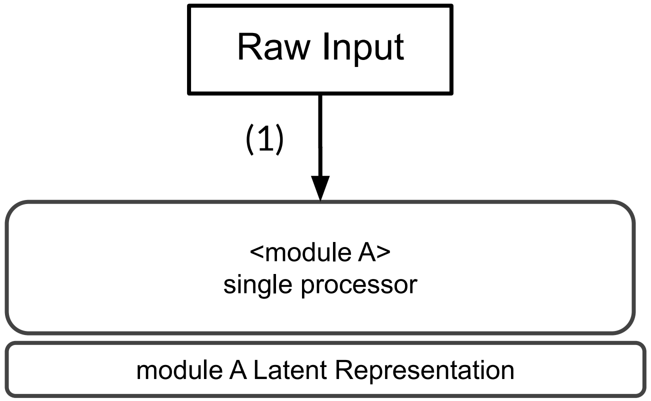

Raw Input
(1)
(2)
(3)
<domain 1, module A> processor
<domain 2, module A> processor
<domain 3, module A> processor
module A Latent Representation

MPCL modules don't forget but there is no easy way to know where to route the raw input between (1), (2) and (3).

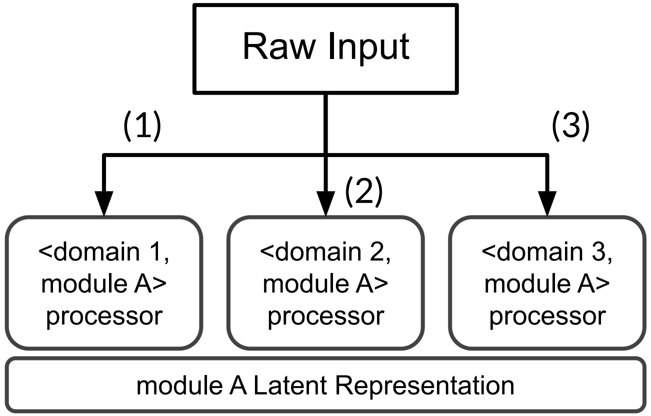
As for knowledge mutualisation, refer to soft-parameter sharing.

Inside Modules: The Routing Problem

Conventional Continual Learning

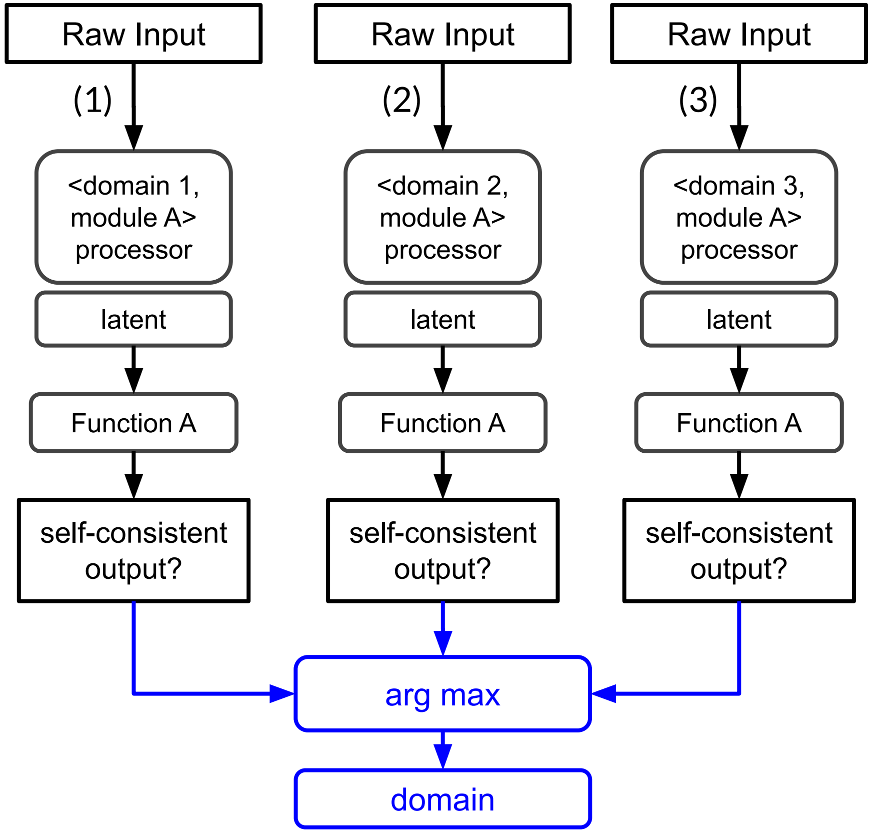


MPCL


Raw Input
(1)
(2)
(3)
<domain 1, module A> processor
<domain 2, module A> processor
<domain 3, module A> processor
module A Latent Representation

MPCL is not strongly committed to one way or the other, but the multi-processor approach makes it easier to define generalization, abstraction and transferability.

Proto-MPCL



Proto-MPCL is this building block.

For MPCL to work at a larger scale, we need proto-MPCL to do quite well on isolated continual-learning tasks.

On paper, it doesn't need to be perfect because the more proto-MPCL blocks you combine to realize various goals, the easier it gets to find ways of detecting inconsistencies between the functions' outputs.

Incremental Domain Generalization

Incremental Domain Generalization

- Both continual learning and domain generalization (DG) techniques try to create rich representations in a non-i.i.d. setting.

- DG can be seen as a special case of domain IL wherein most of the hard work is done *before* observing out-of-distribution data.

- One of the most promising approaches to DG is causal representation learning. The idea of uncovering the causal structure of the problem is similar to MPCL's meaning alignment.

- MPCL is inverse DG. Instead of building high-quality abstraction hypotheses that are meant to generalize well to unknown domains from the get go, MPCL tries out hypotheses over many domains to assess their generalization power.

- If they pass the quality test, they are entrusted with stronger, longer-lasting connectivity with other modules.

Incremental Domain Generalization

- According to Rule 6, “if a module cannot accommodate new domains, it should be removed.”

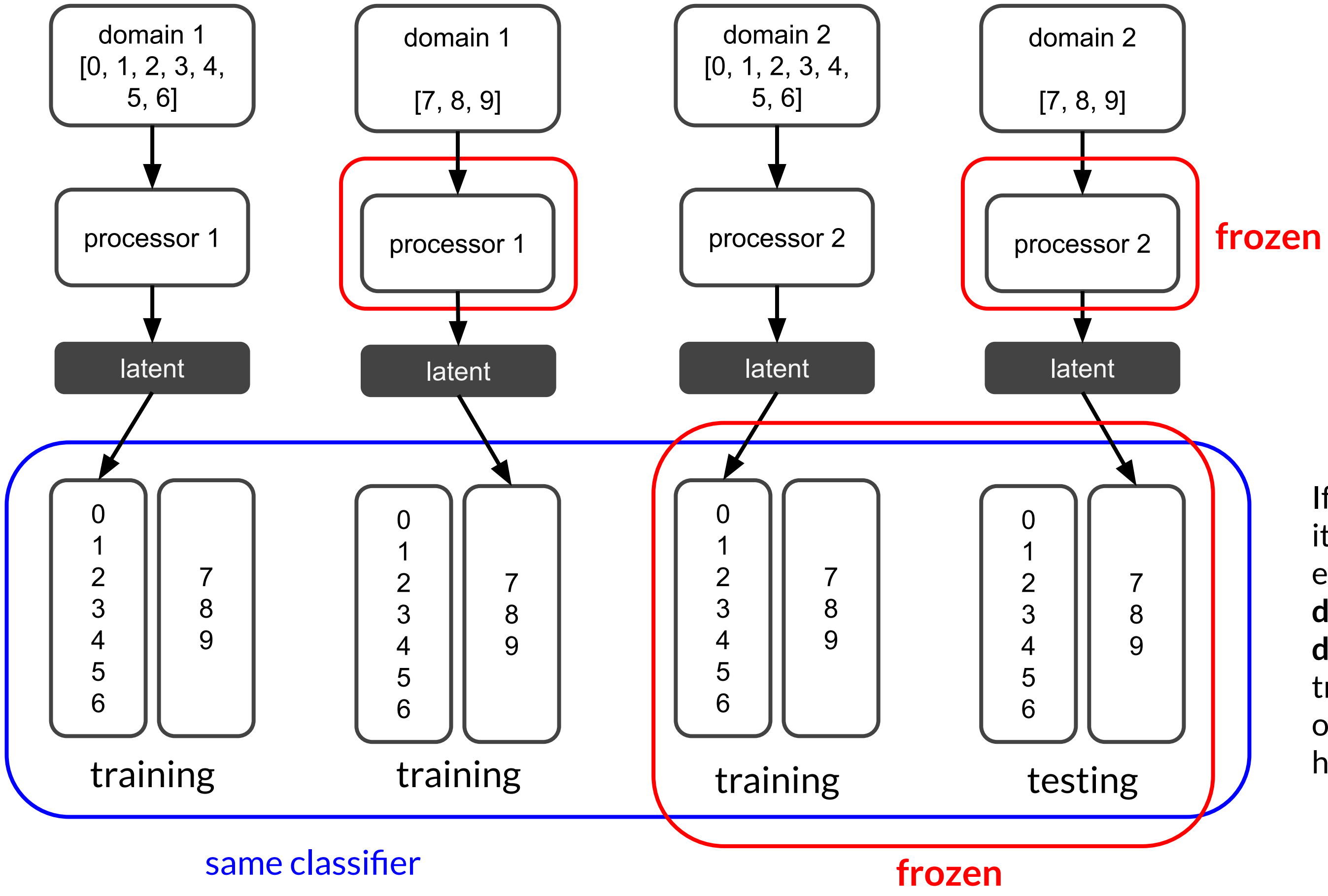
- Rule 6 begs the question of how modules can fail. Processors are coerced to compute latent units that make correct predictions. It is unlikely that they fail if they are given enough capacity and degrees of freedom.

- To alleviate this problem, one can create two functions mapping latent units to predictions. Both are optimized on the first domain. Only one of the two functions is optimized on the other domains. The second function serves as a way to measure whether “the module can accommodate new domains”.

- For now, we will consider the case of dividing existing functions into two parts, e.g. [classifying digits 0, 1, 2, 3, 4, 5, 6], [classifying digits 7, 8, 9].

- Later, we might consider auxiliary functions or connected modules to detect misalignments.

Incremental Domain Generalization



If **domain 2** fails on [7, 8, 9], it is because the reasoning employed to predict [7, 8, 9] on **domain 1** doesn't transfer to **domain 2**. The latent layer trained from **domain 1** turned out to be a poor abstraction hypothesis.

Incremental Domain Generalization

- To summarize this section, latent units align with our reality if one can successfully reason over them across a wide range of domains, with limited fine-tuning on these domains.

- Future work: try to solve this problem with causal representation learning or meta-learning.

Meaning

Meaning != Representation

I mean “representation” in the ML sense, as in “representation learning”.
I do not mean “mental representation”.

If Representation-Preserving Continual Learning (RPCL) were a thing, it would not be the same thing as MPCL. It would be more limited and more limiting.

- Not every representation vector needs stability, whereas meaning always needs stability ;
- Representation vectors can be more fine-grained than meaning.

Meaning != Representation

- In a classification setting, one could argue that meaning is redundant as it is conceivable to enumerate all the representation vectors that need stability and call it a day.

- In regression and motor settings, however, it becomes harder to identify the representations that need stability*, how to measure it and whether some latent values need more stability than others. It is useful to look at the problem from the meaning angle, i.e. the link between representations and goals. Goal-realizing functions need to remain *accurate*. They need not remain *stable* at all times.

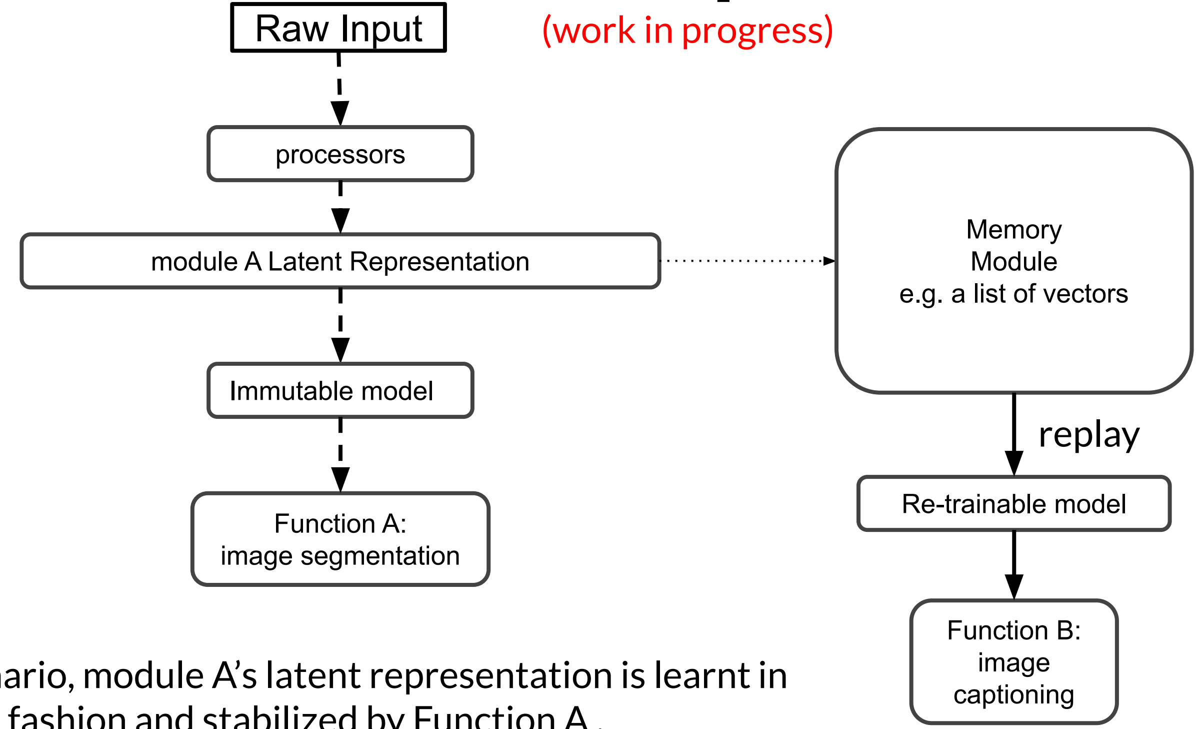
* I'm still debating whether it would be practical or not to require stability for every single representation vector of the training set, thereby doing away with meaning.

Real Examples

(work in progress)

Two-module scenario: example

(work in progress)

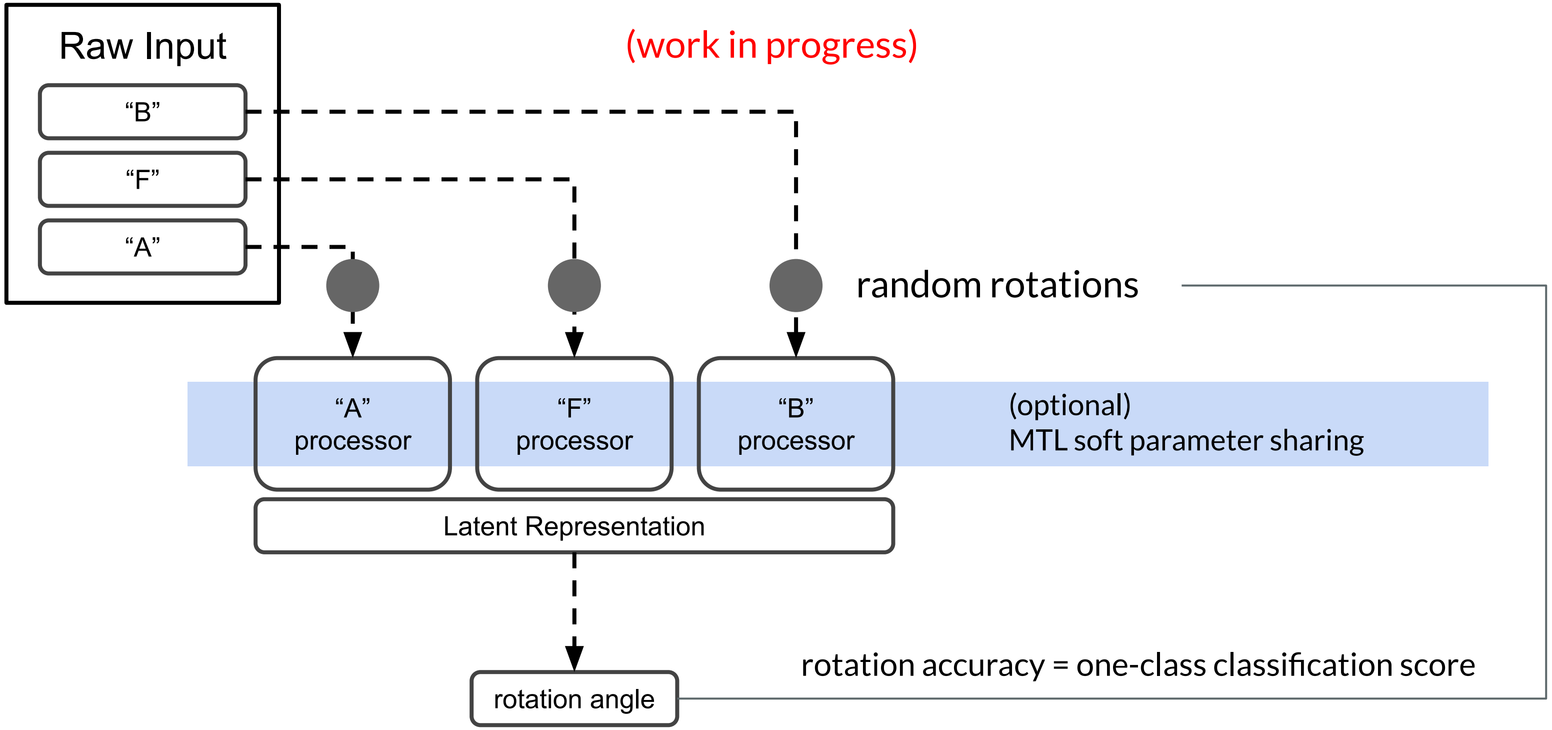


Raw Input
processors
module A Latent Representation
Immutable model
Function A: image segmentation
Memory Module e.g. a list of vectors
replay
Re-trainable model
Function B: image captioning

In this scenario, module A's latent representation is learnt in a continual fashion and stabilized by Function A .

The re-trainable model utilizes A's representations to realize Function B.

One-module scenario: EMNIST



This works because "rotation" is a subjective concept whose meaning is not carried by the input alone. This is why the system works better with a custom processor for each letter, and this is why it is an adequate function for detecting discrepancies.

Example of degenerate module: if the function is to denoise the images or to count the number of black pixels, it can be done without knowing the letter, thus it doesn't help us predict the letter class.