Making It Work

It = Training a deep network

Philipp Krähenbühl, UT Austin

Announcements

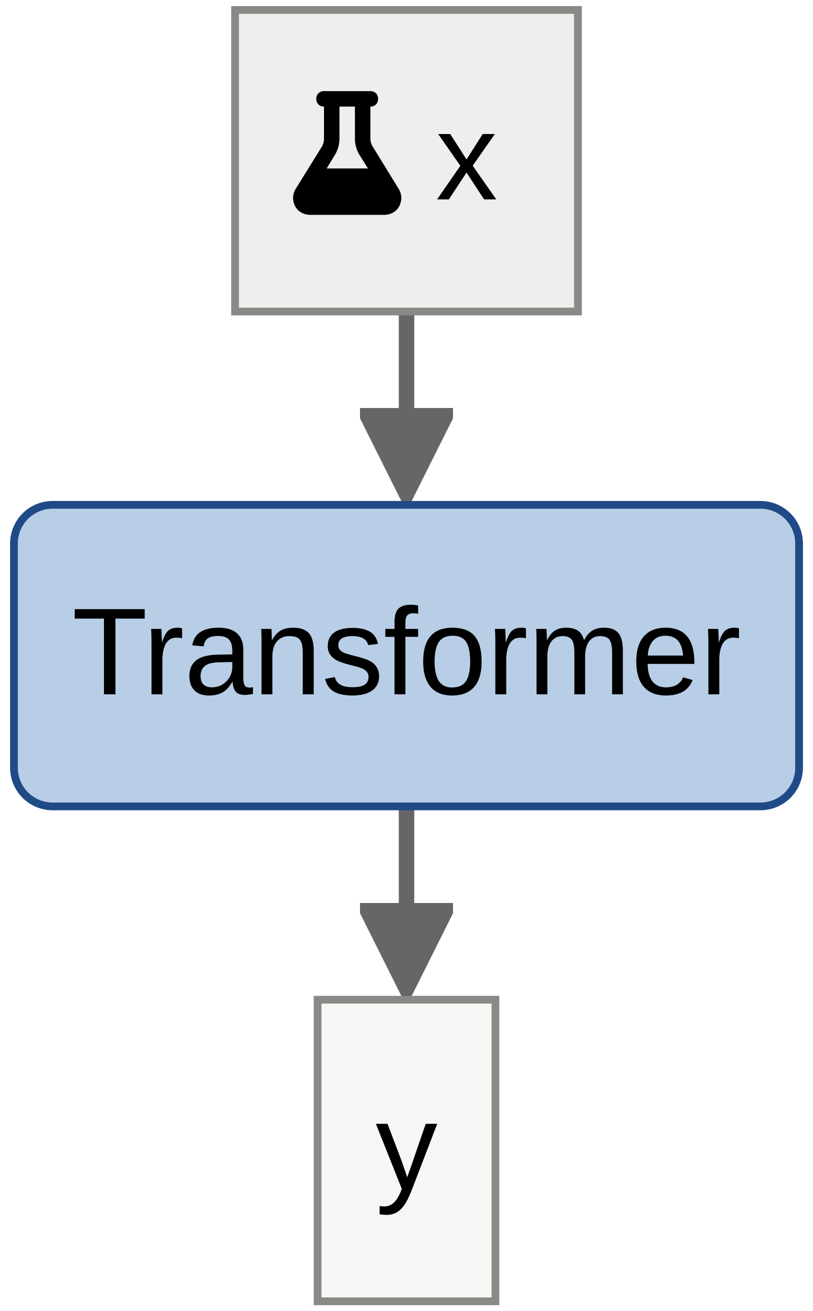
- I’ll be traveling next week (class is optional)
  - TAs will go over HW03 in class
  - Will post a video with the lecture material

(Vision) Transformers in PyTorch

Making it work

Recap

Model



Data

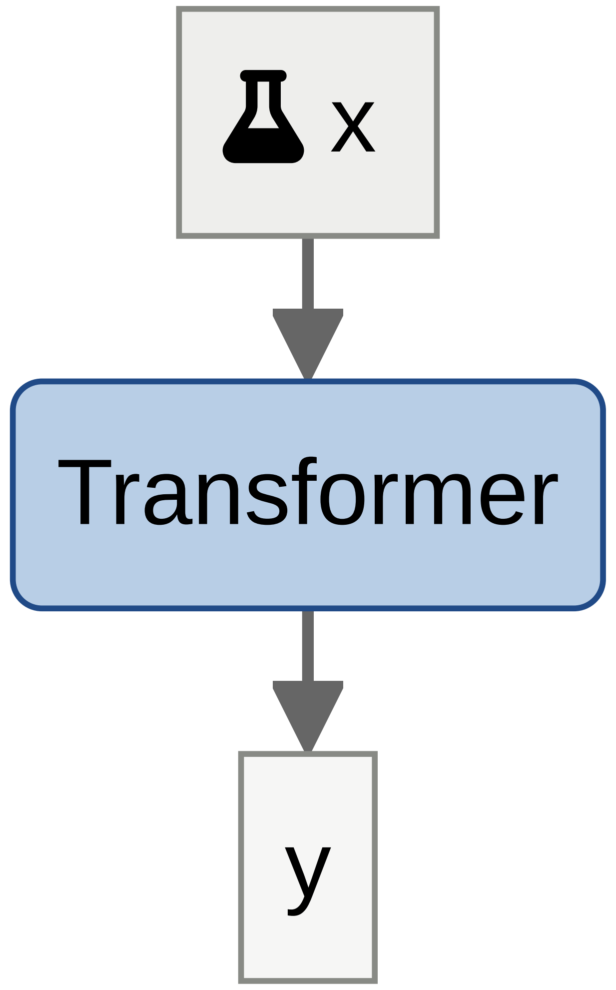
Optimization

```
m = 0
for epoch in range(n):
  for (x, y) in dataset:
    J = ∇l(θ|x,y)
    m = J + momentum * m
    θ = θ - ϵ * m.mT
```

How to train a network?

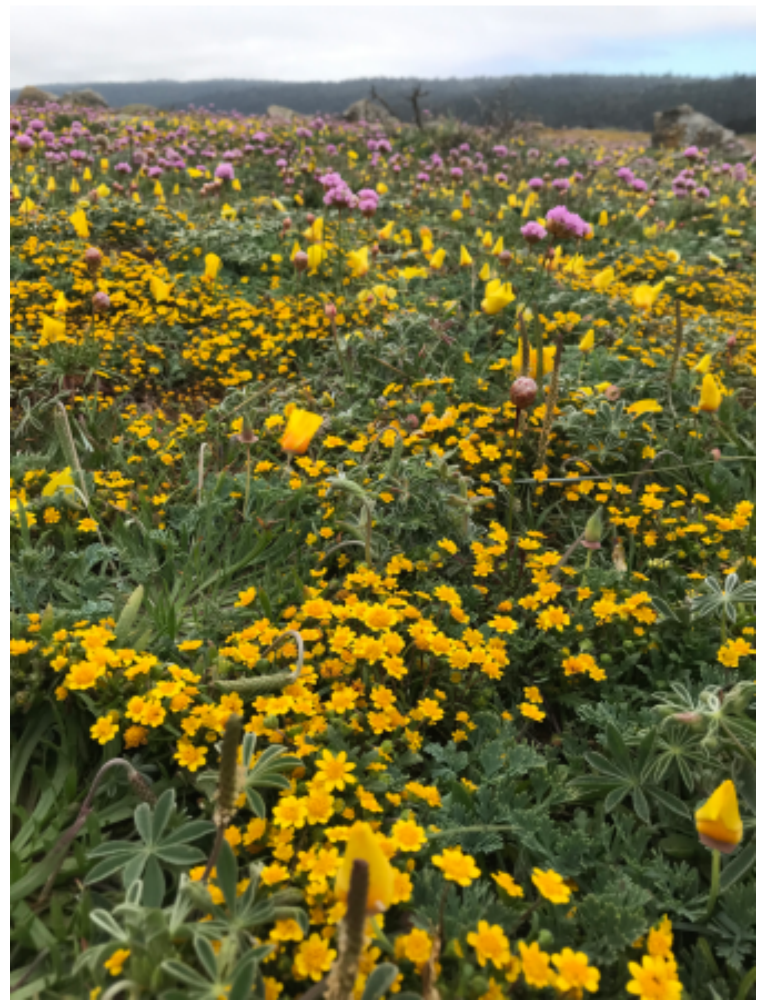
Collect Data

Design / download architecture



Train model

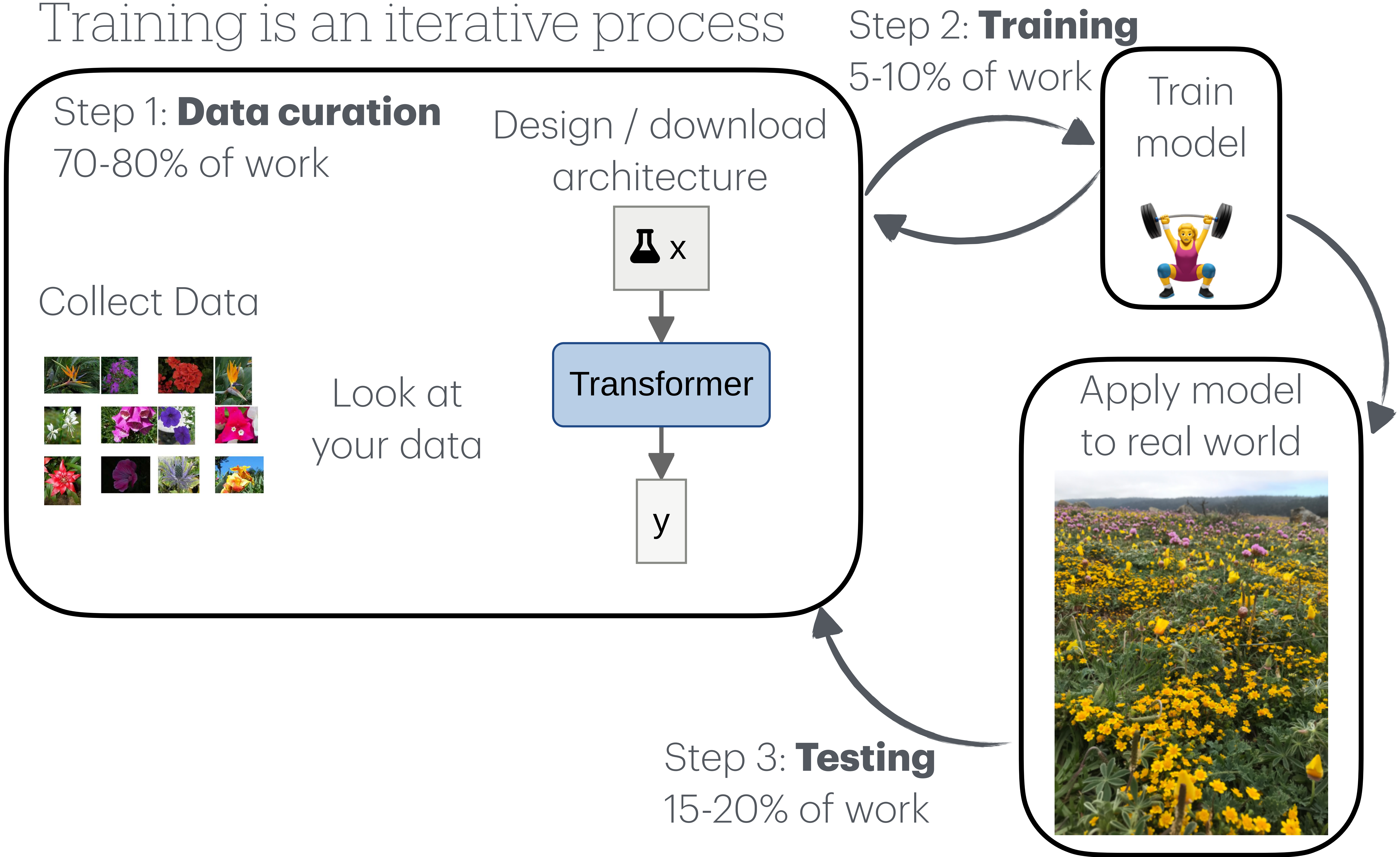
Apply model to real world



This never works !!!

How to train a network?

Training is an iterative process



Data Curation

Looking at Your Data

Images

- Randomly sample
- Smallest / largest file size
- Rare classes

Try solving the task manually

Random Images







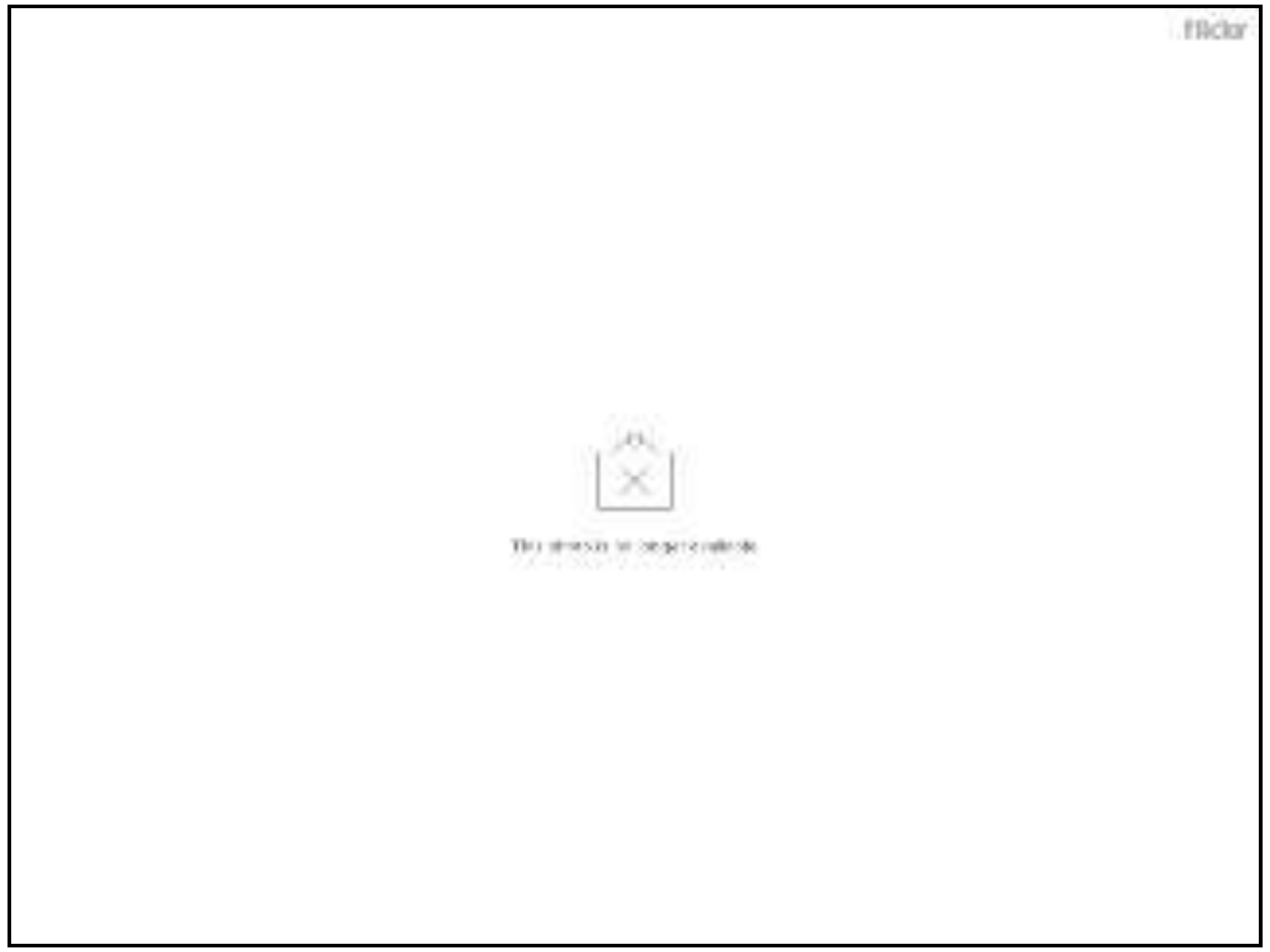




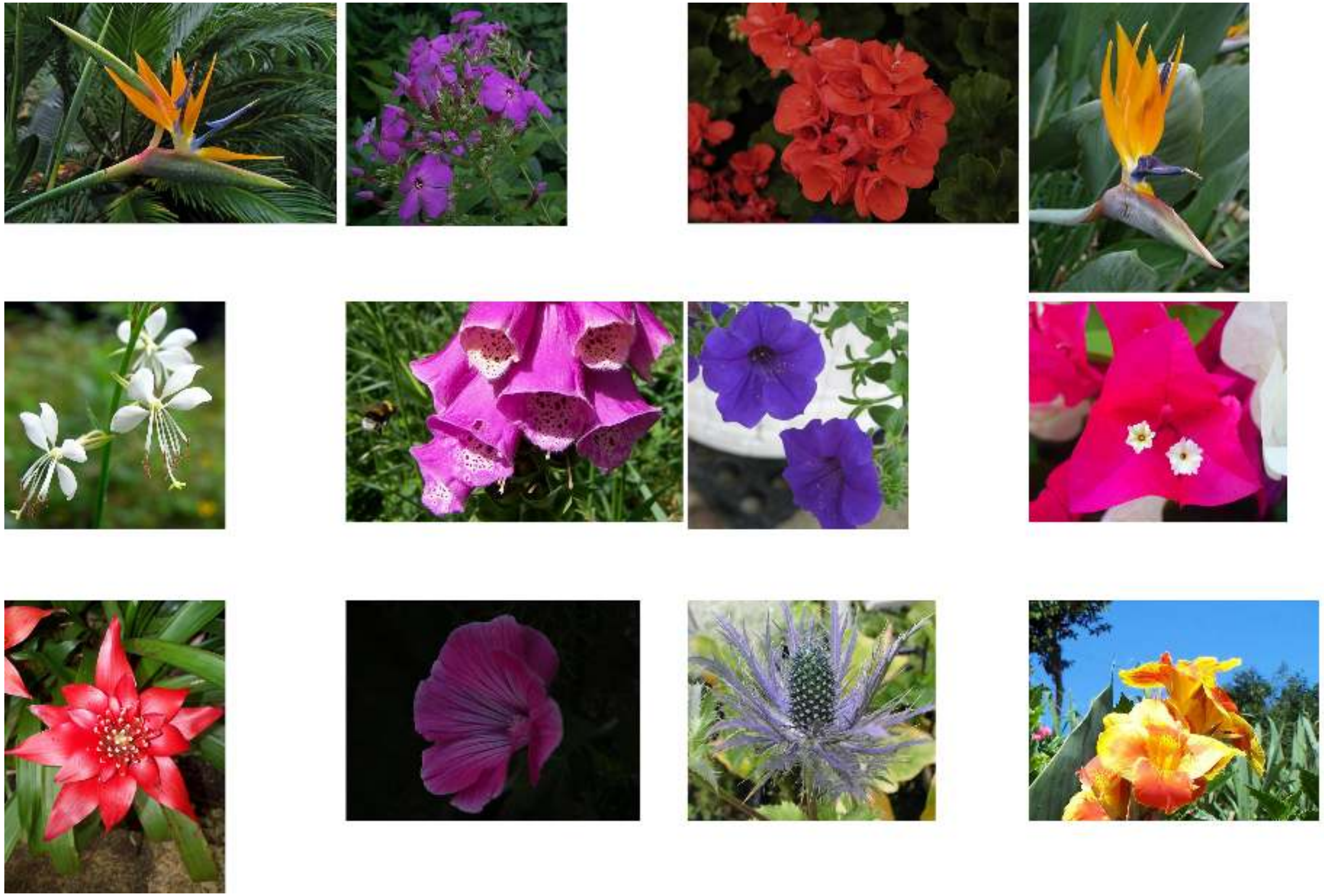

Largest File Size

Smallest File Size

Smallest File Size

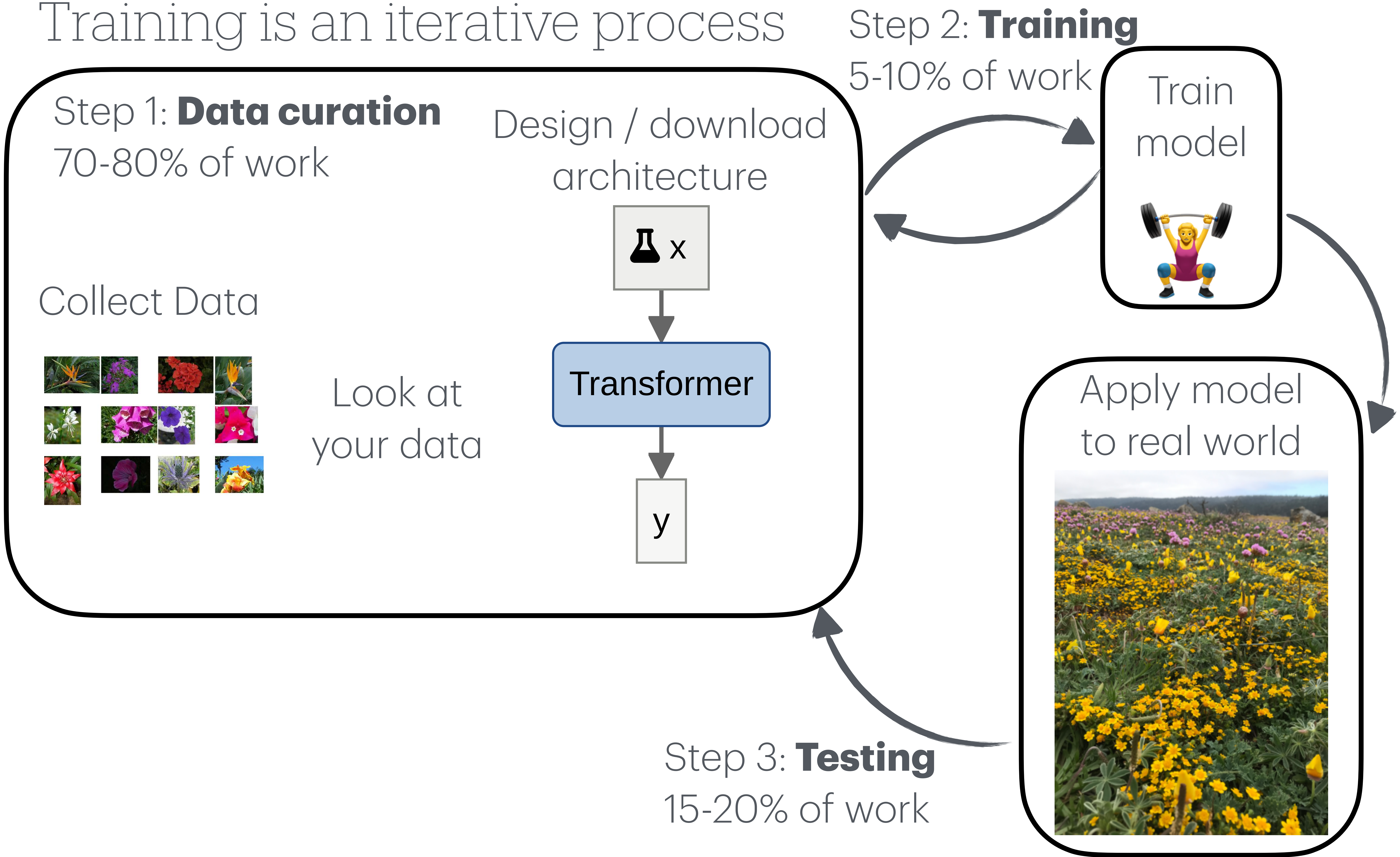


Solving the Task Manually



How to train a network?

Training is an iterative process



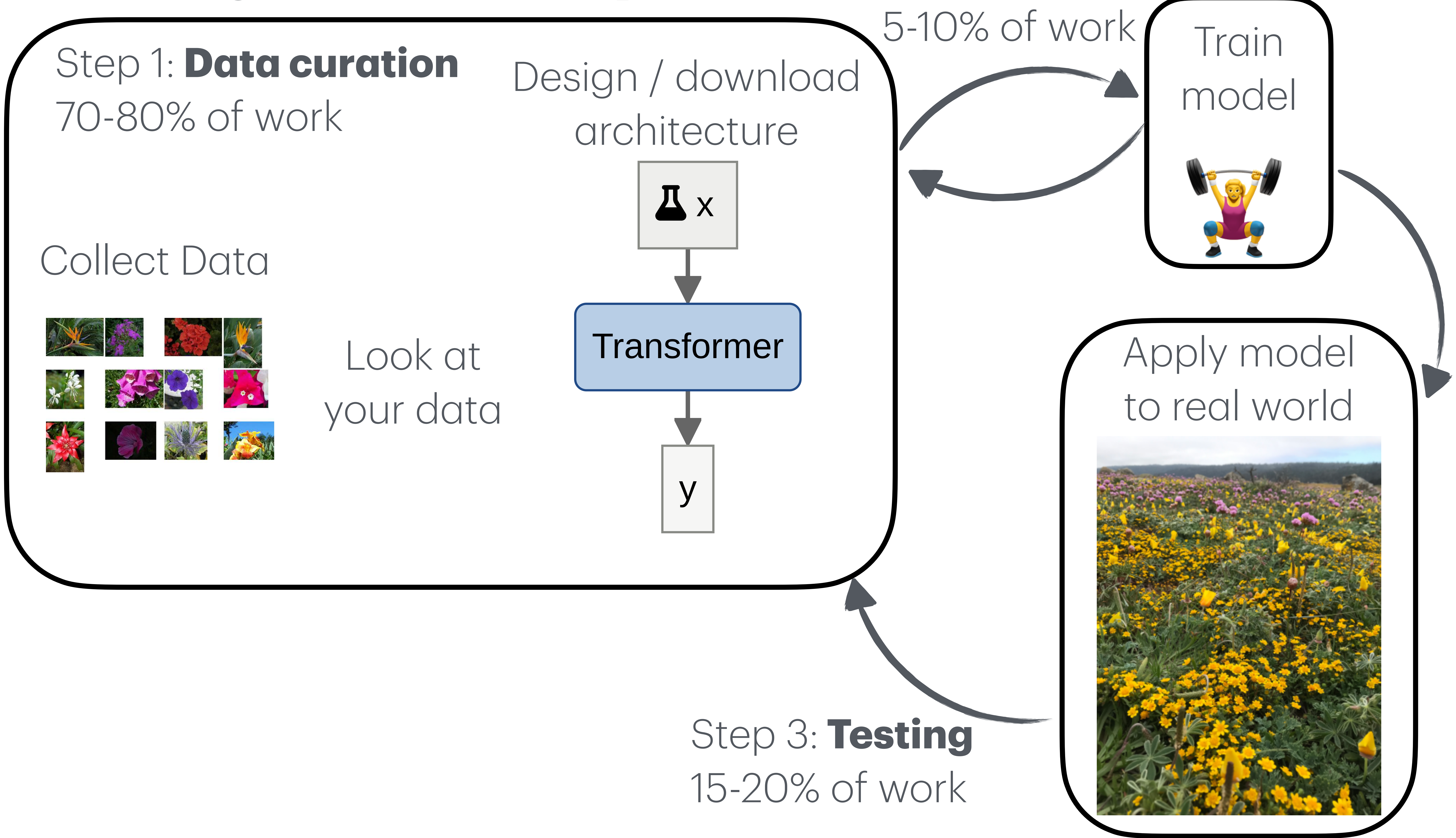
More on data Next Week (video)

Training - Next Week (video)

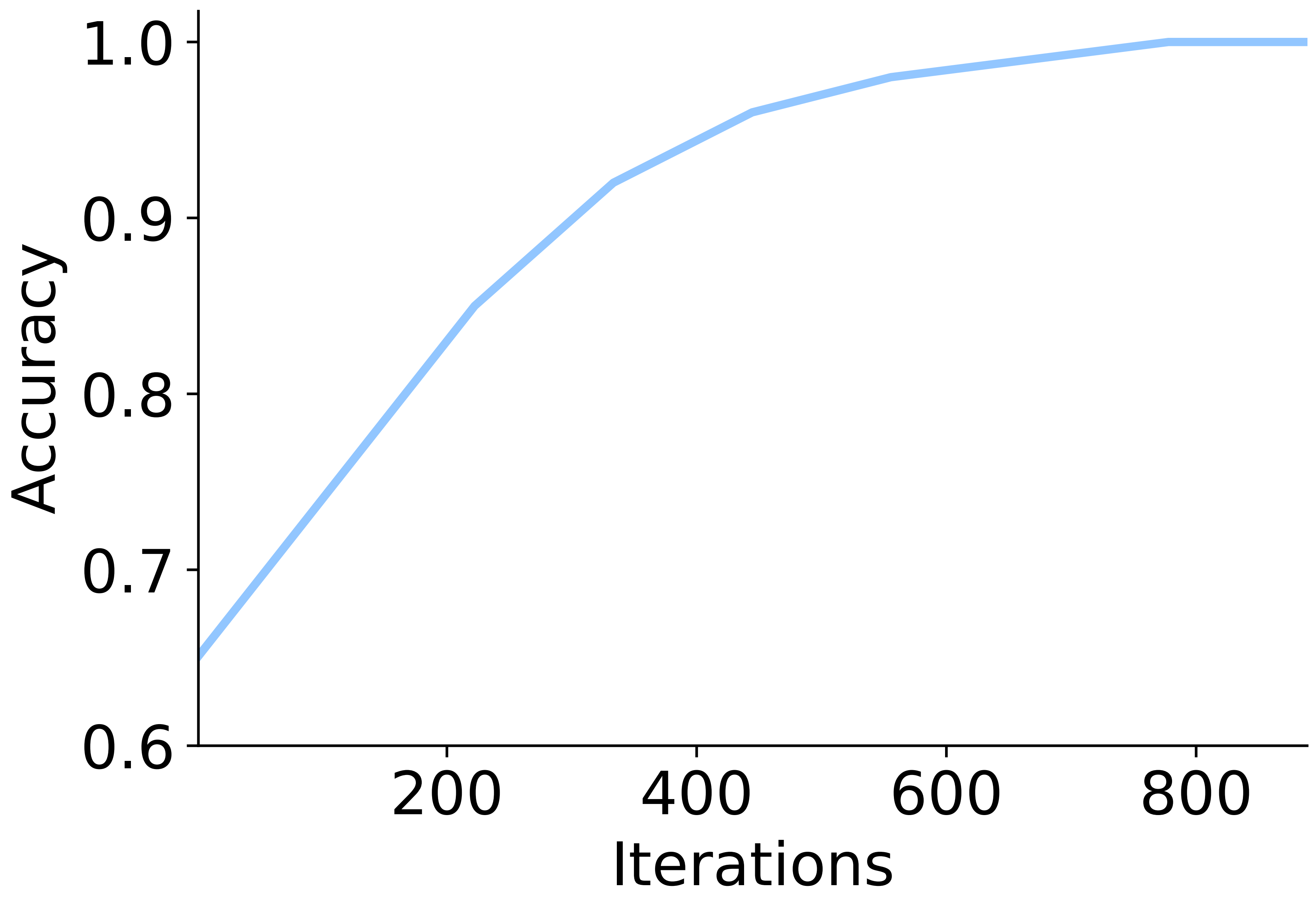
Testing

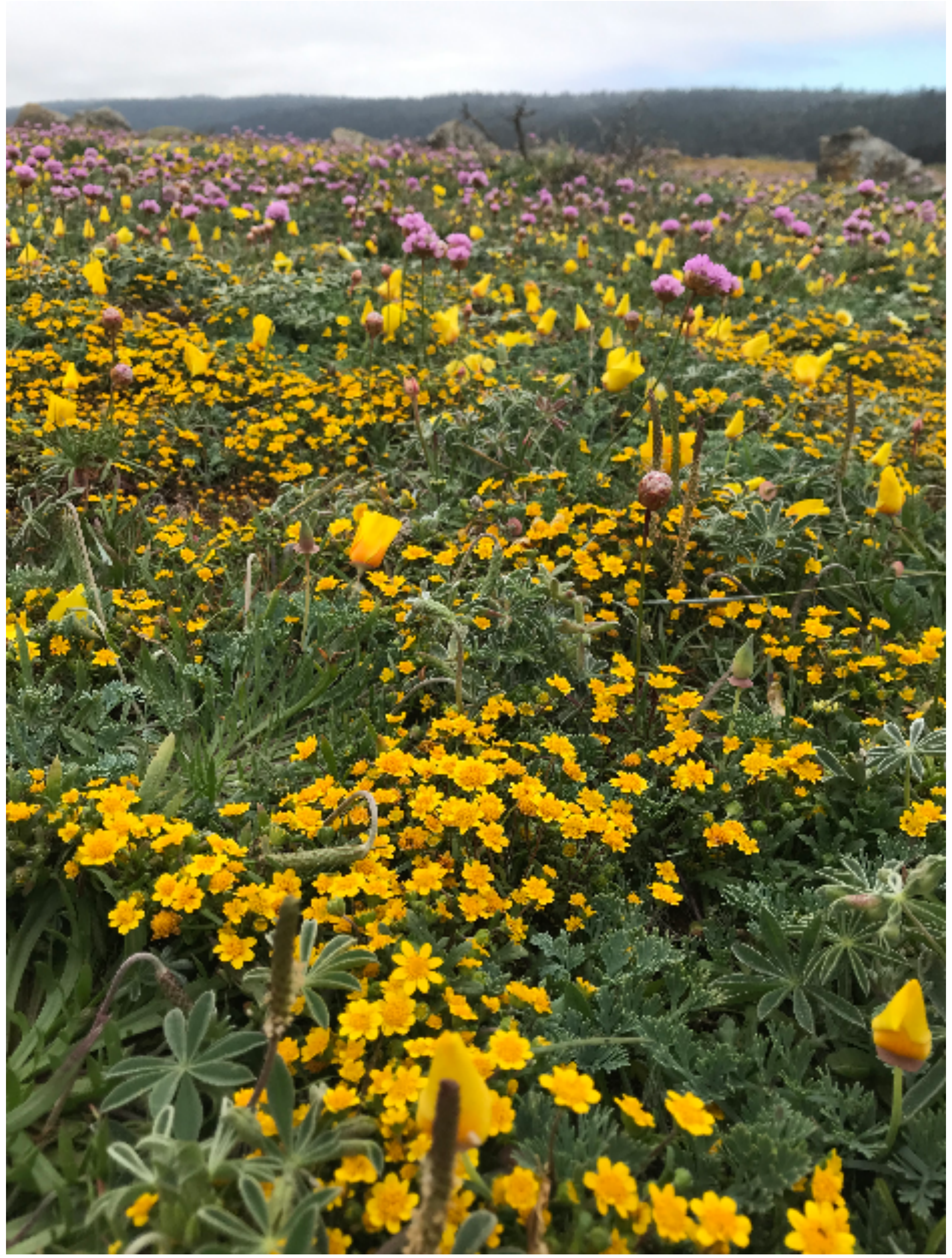
How to train a network?

Training is an iterative process



Perfect Training Accuracy Achieved





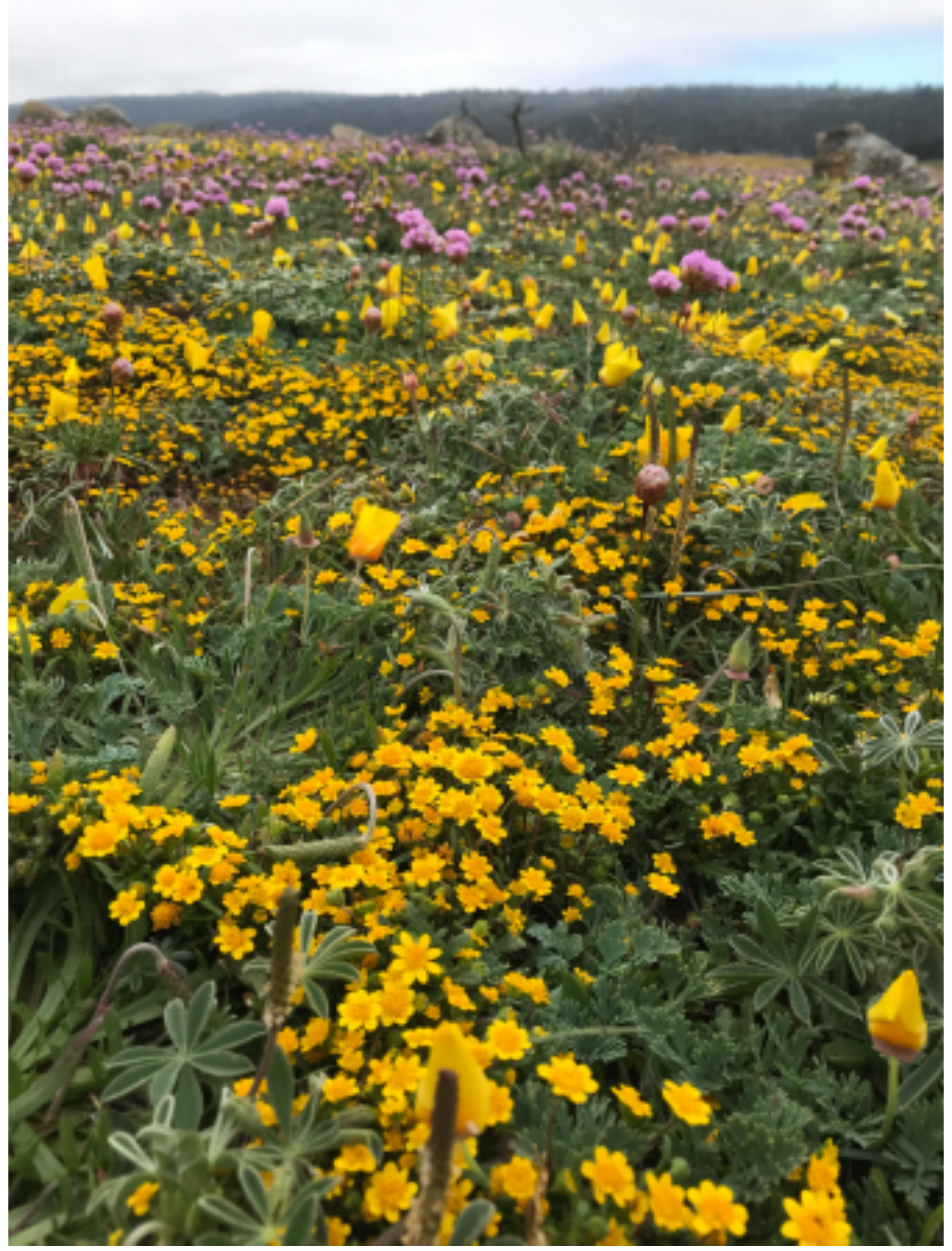
Our Data Is a Proxy for the Real World

Optimization Objective

Learn a model that works well on our dataset

Goal

Learn a model that works well in the real world



Dataset Splits

Training set

Learn model parameters

Validation set

Learn hyper-parameters

Test set

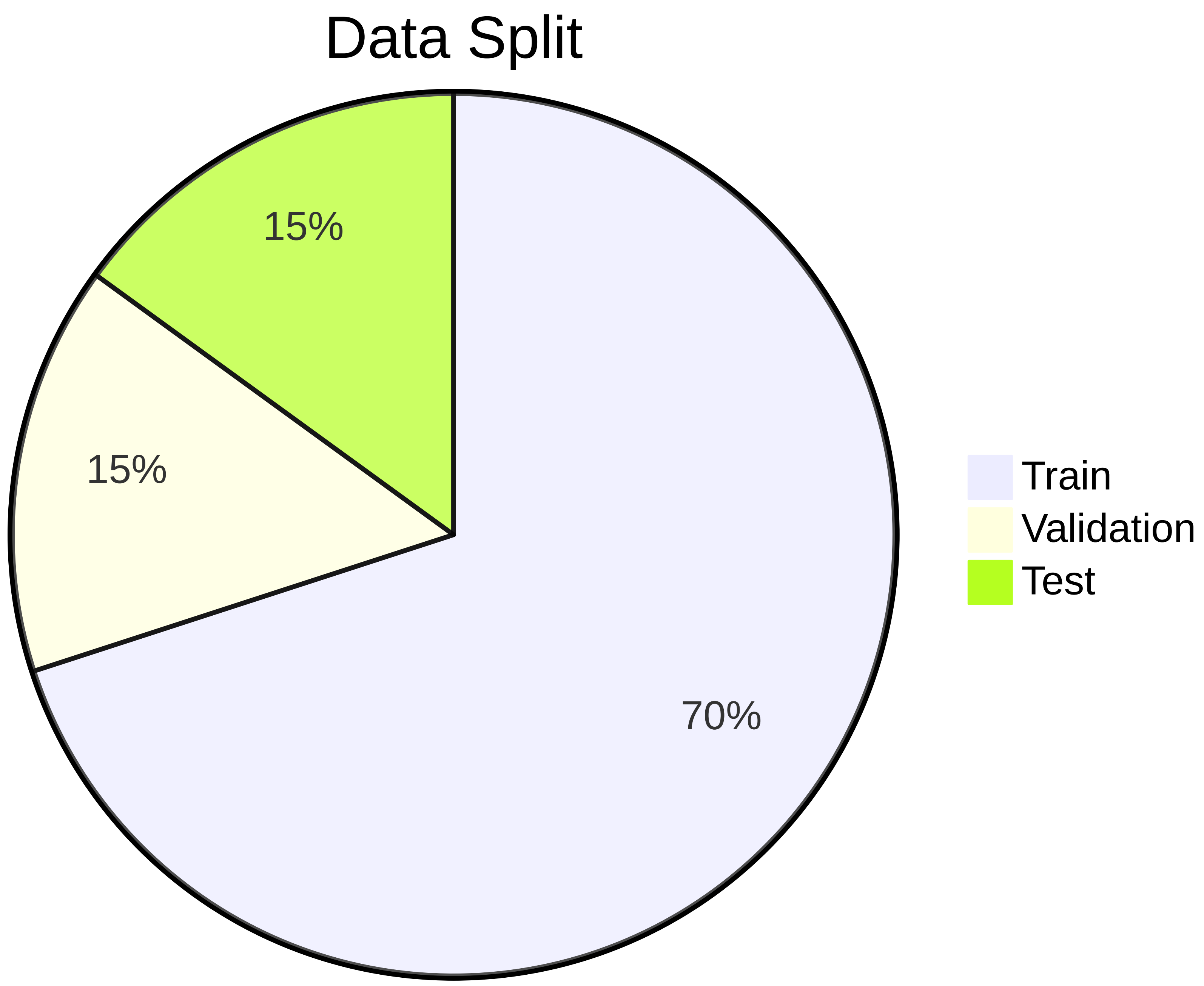
Measure real world performance

Training Set

Used to train all parameters of the model

Model will work very well on training set

Size: 60-80% of data

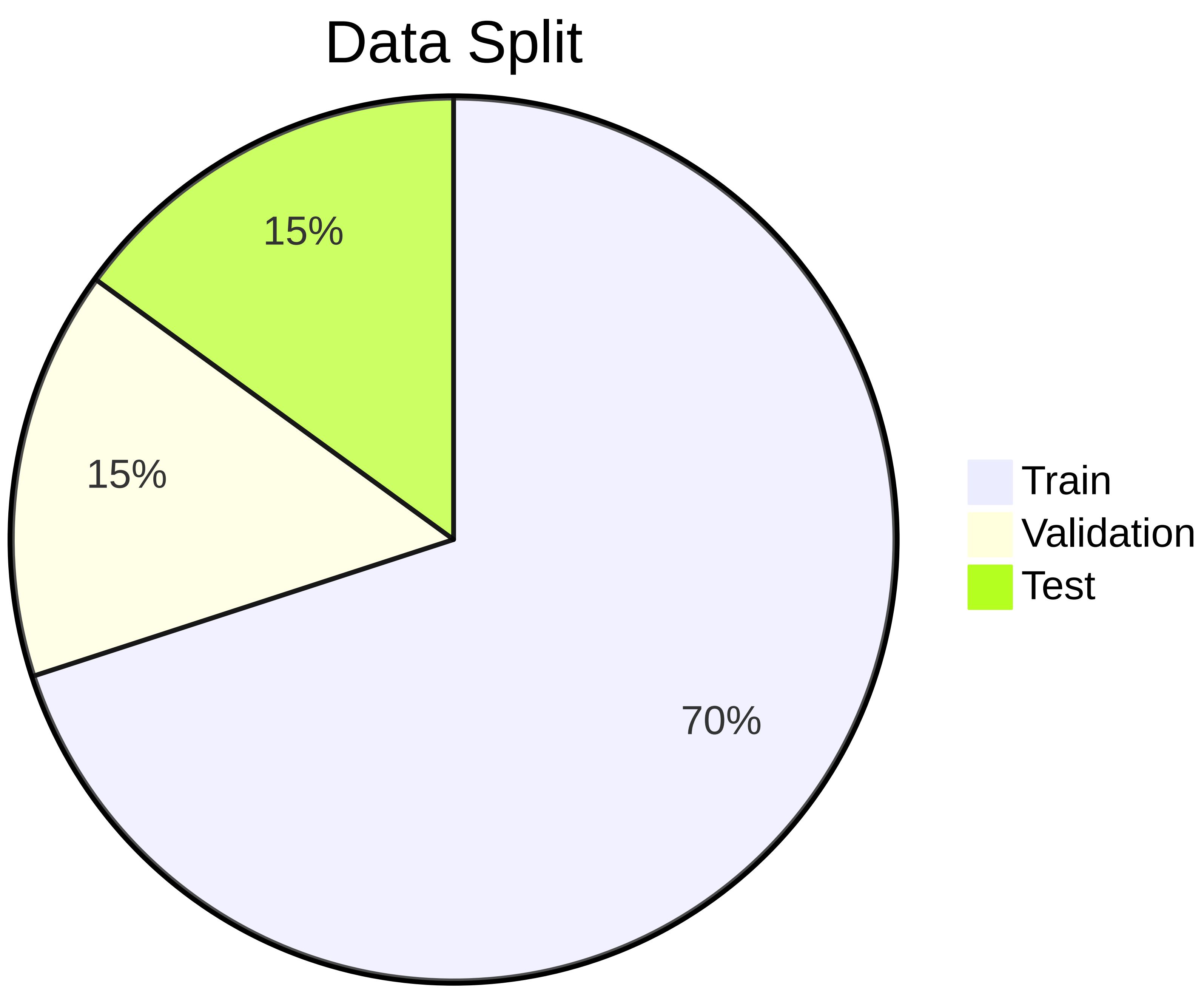


Validation Set

Used to determine how well the model works

Used to tune model and hyper-parameters

Size: 10-20% of data

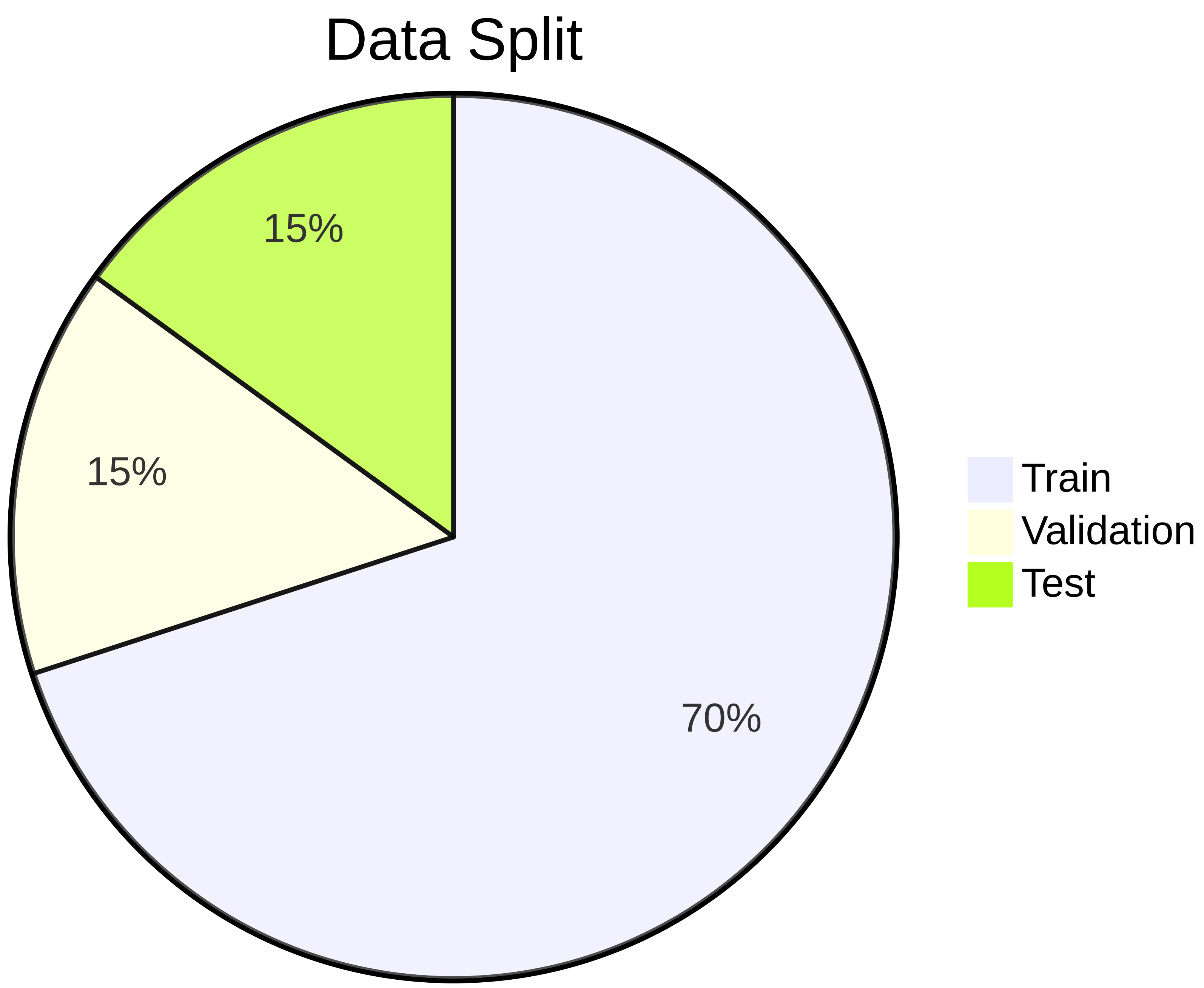


Testing Set

Used to measure model performance on unseen data

Used exactly once

Size: 10-20% of data

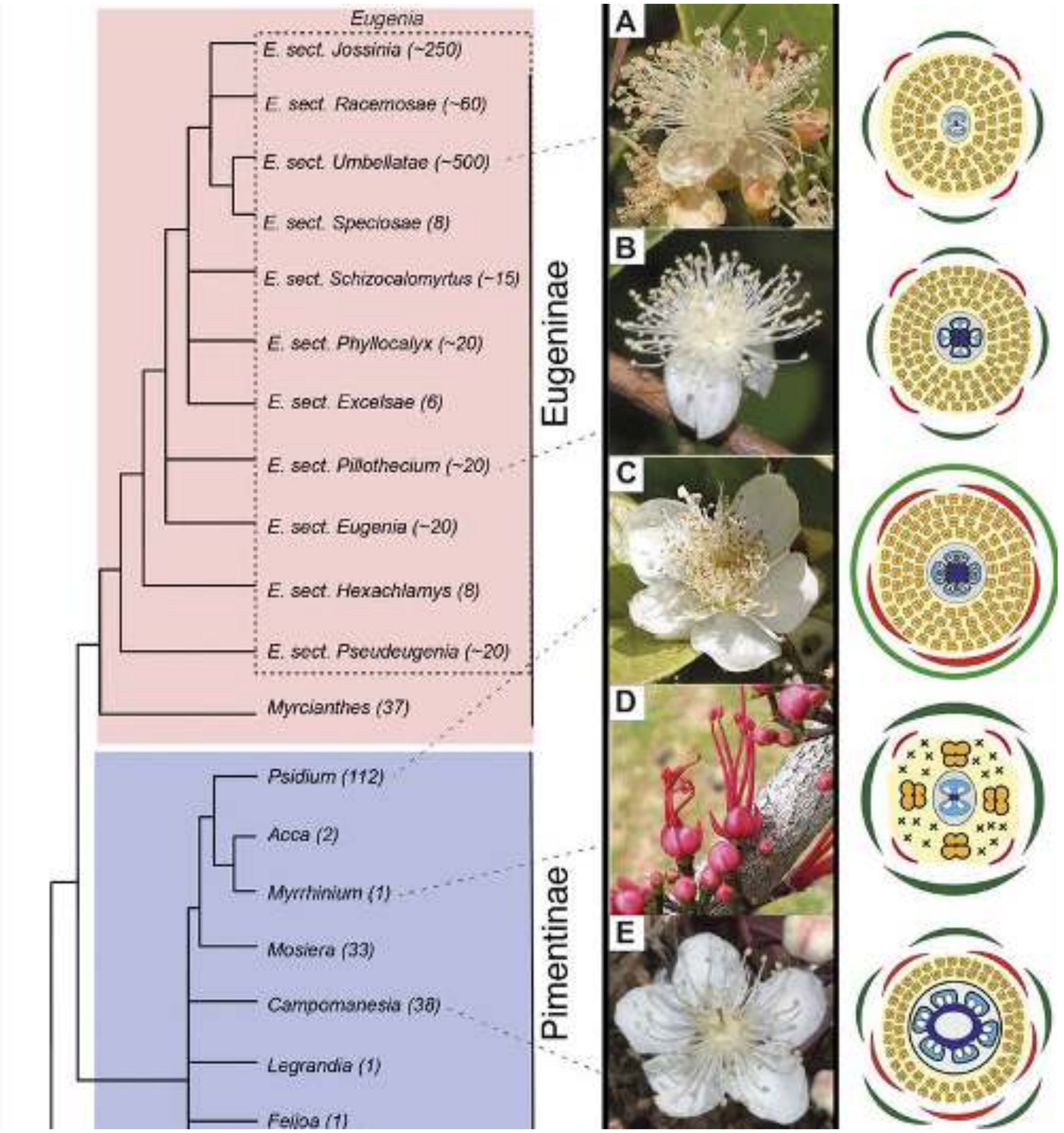


How Do We Split the Data?

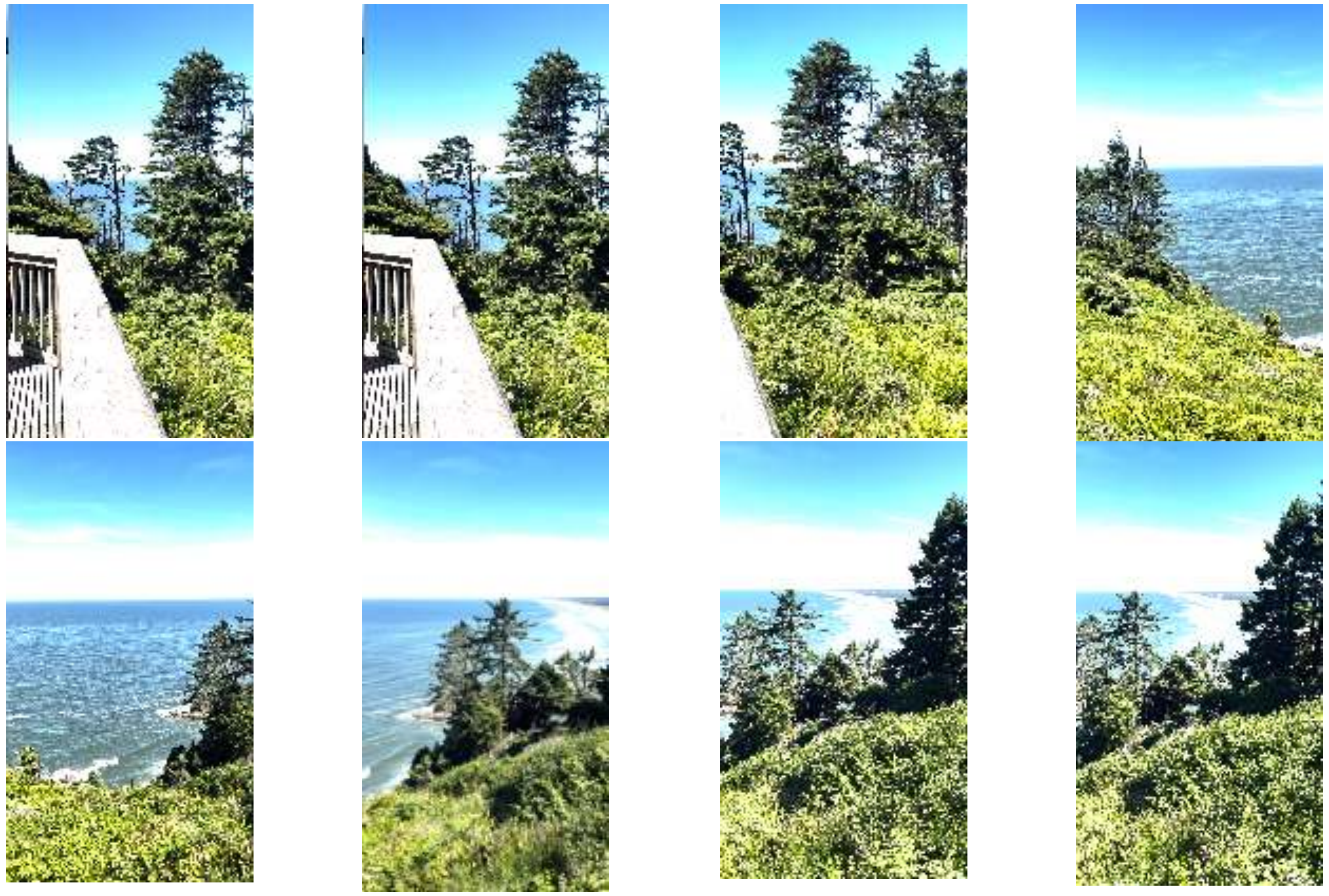
Random Sampling Without Replacement

Warning: Correlated Data

Flowers in a genus[1]:



Images in a Video



1. Vasconcelos et al. A Systematic Overview of the Floral Diversity in Myrteae (Myrtaceae). Systematic Botany 2019. ↩

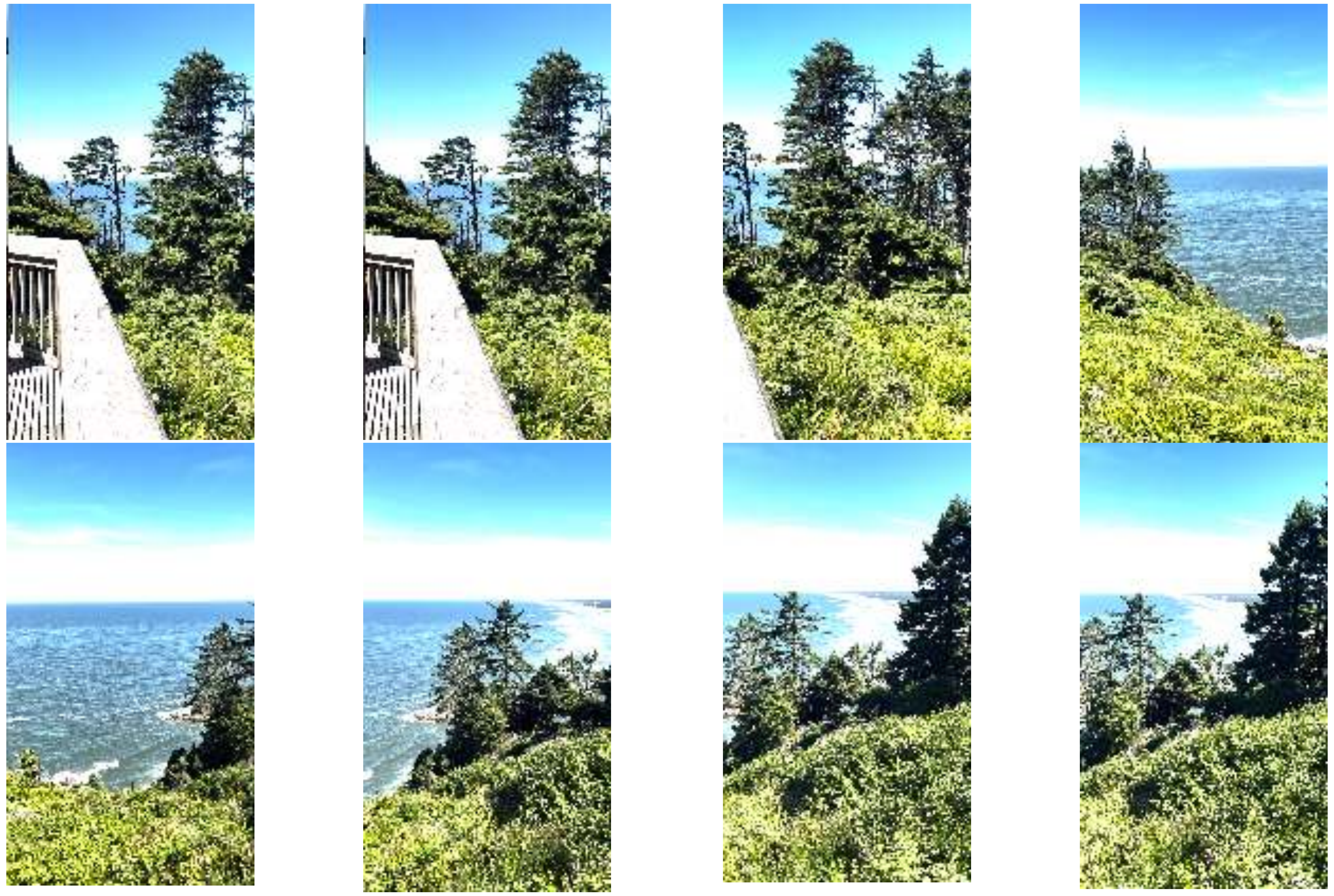
Is Correlated Data Always Bad?

Correlated Data Is Bad When

- model should generalize outside the correlated data

Correlated Data Is Good When

- model should perform well on the correlated data
- e.g. auto-labeling system



Dataset Splits

Training set

Learn model parameters

Validation set

Learn hyper-parameters

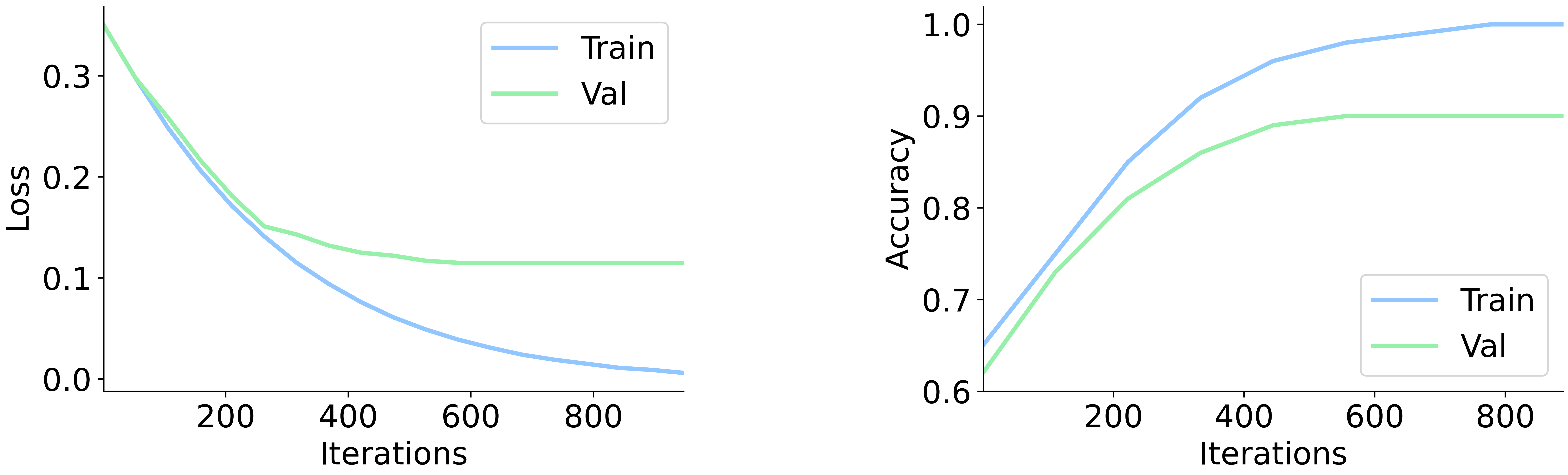
Test set

Measure real world performance

Overfitting

$$L(\theta|\mathcal{D}_{train}) \ll L(\theta|\mathcal{D}_{val})$$

$$\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}_{train}}[l(\theta|\mathbf{x},\mathbf{y})] \ll \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}_{val}}[l(\theta|\mathbf{x},\mathbf{y})]$$

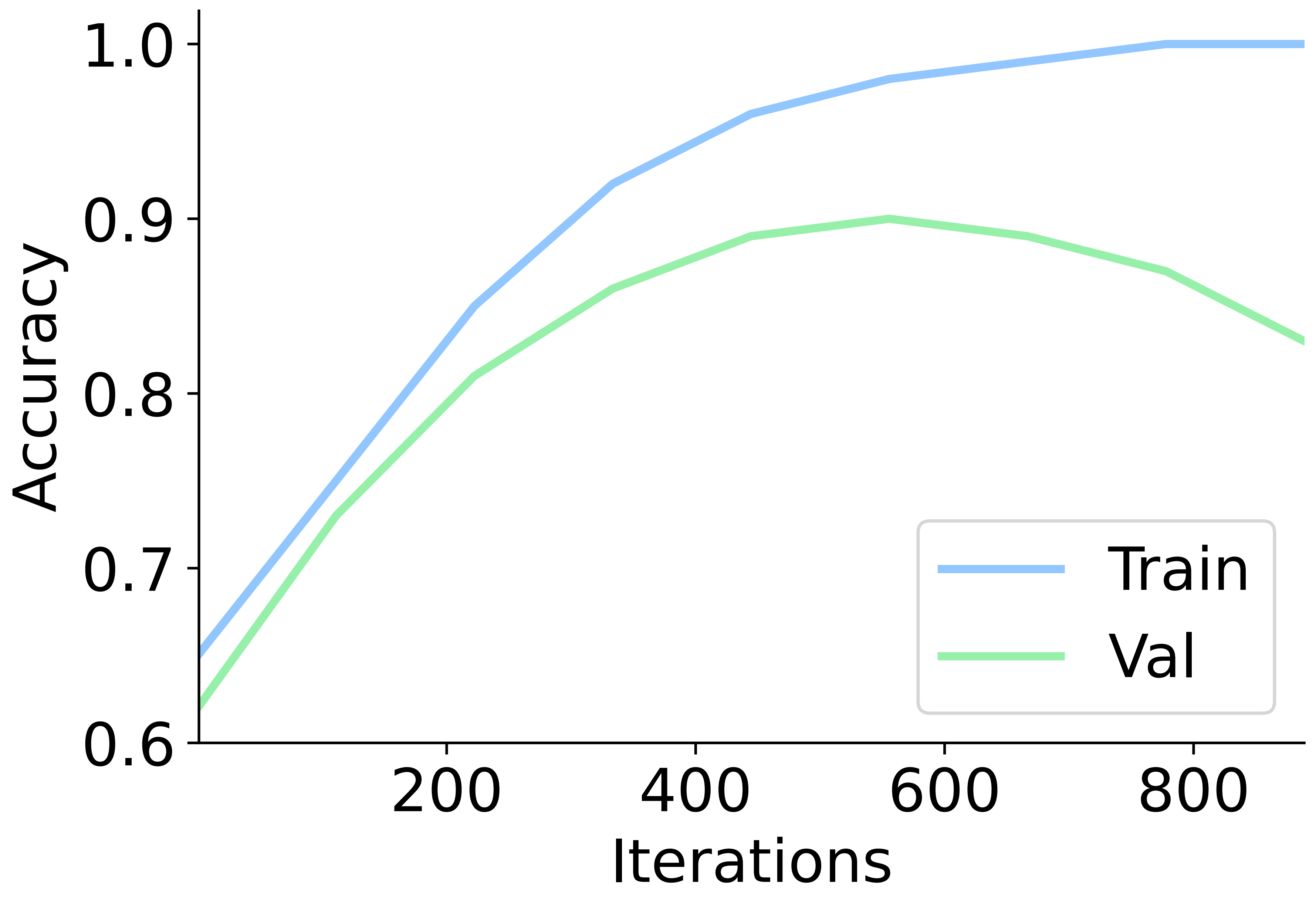


Detect overfitting with data splits

Validation set checks overfitting of **parameters** θ

Test set checks overfitting of **hyper-parameters**

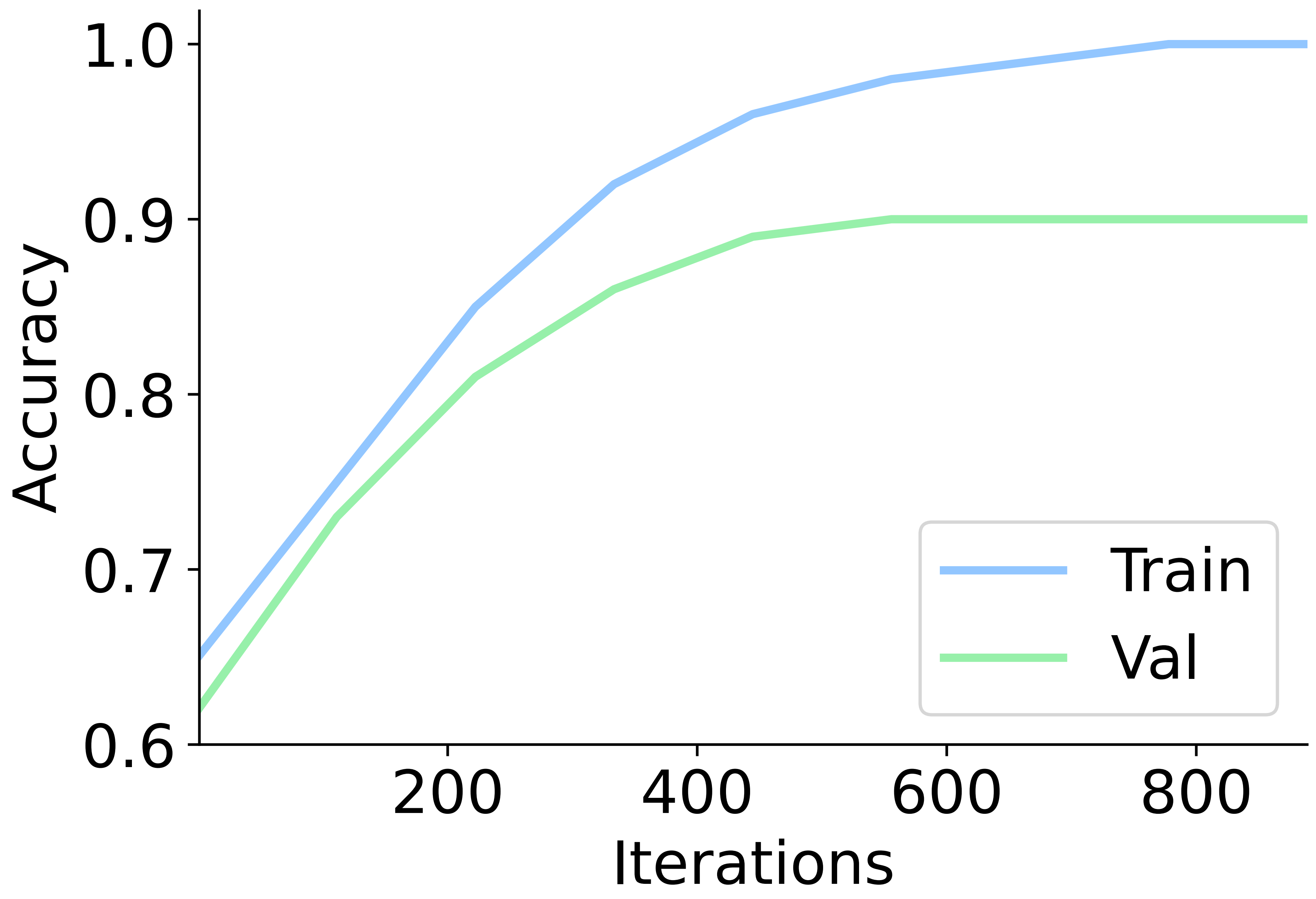
- i.e. number of layers, dimensions



Is overfitting always bad?

Not Really

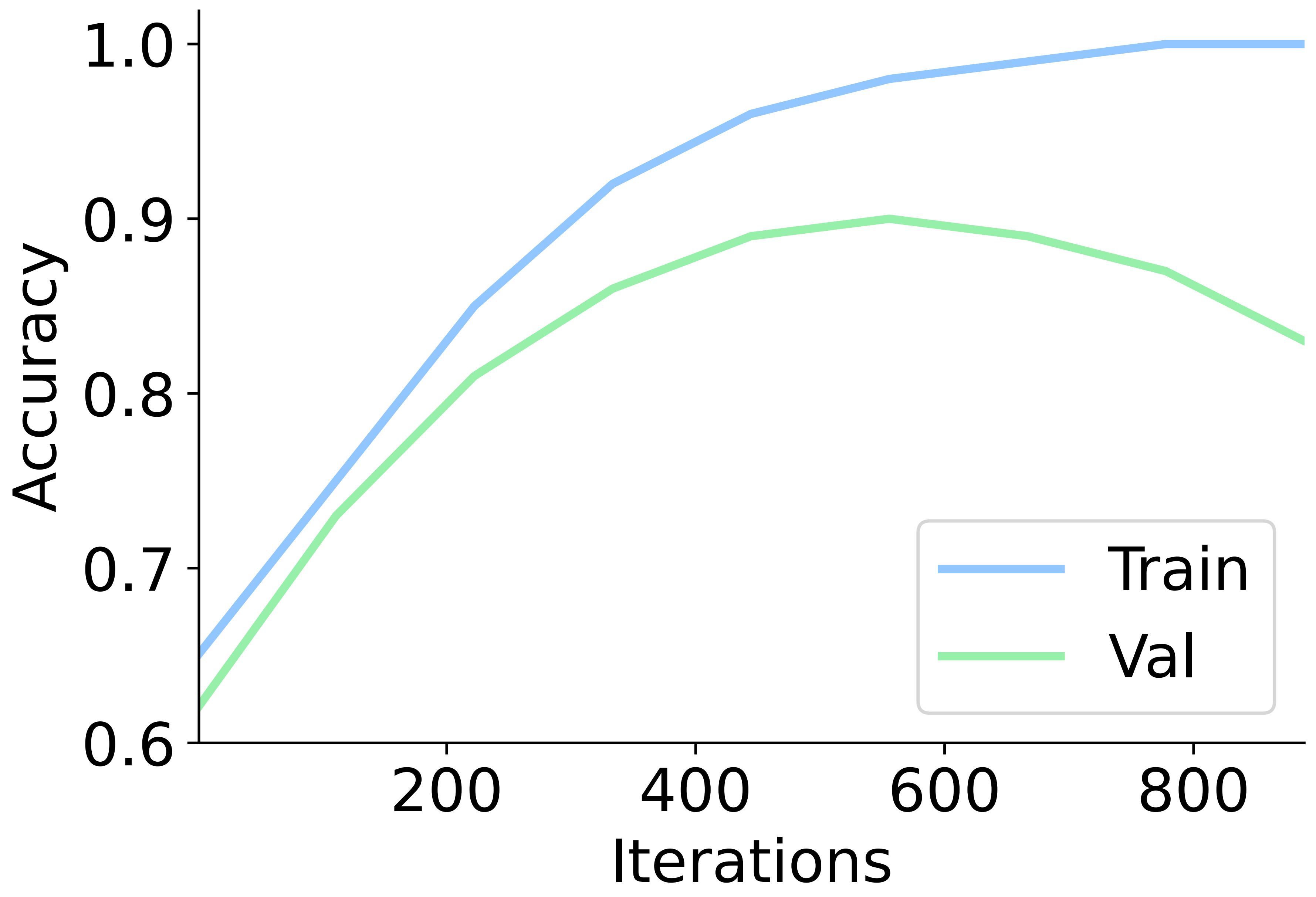
- Only bad if the validation performance decreases



Why do we overfit?

Sampling bias

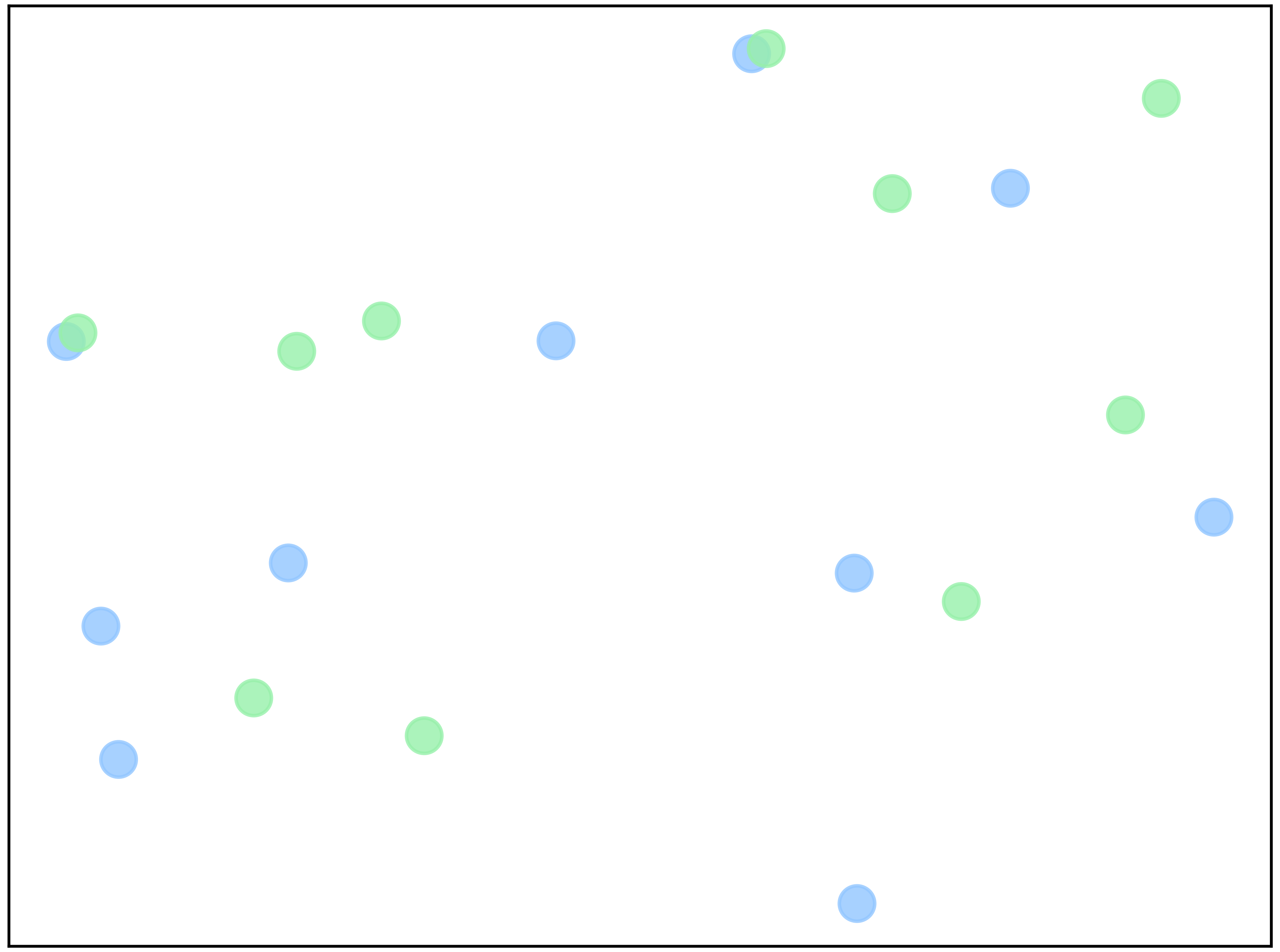
- Fitting patterns that exist only in train set
- Gradients from the same data points multiple times



Why Do We Overfit?

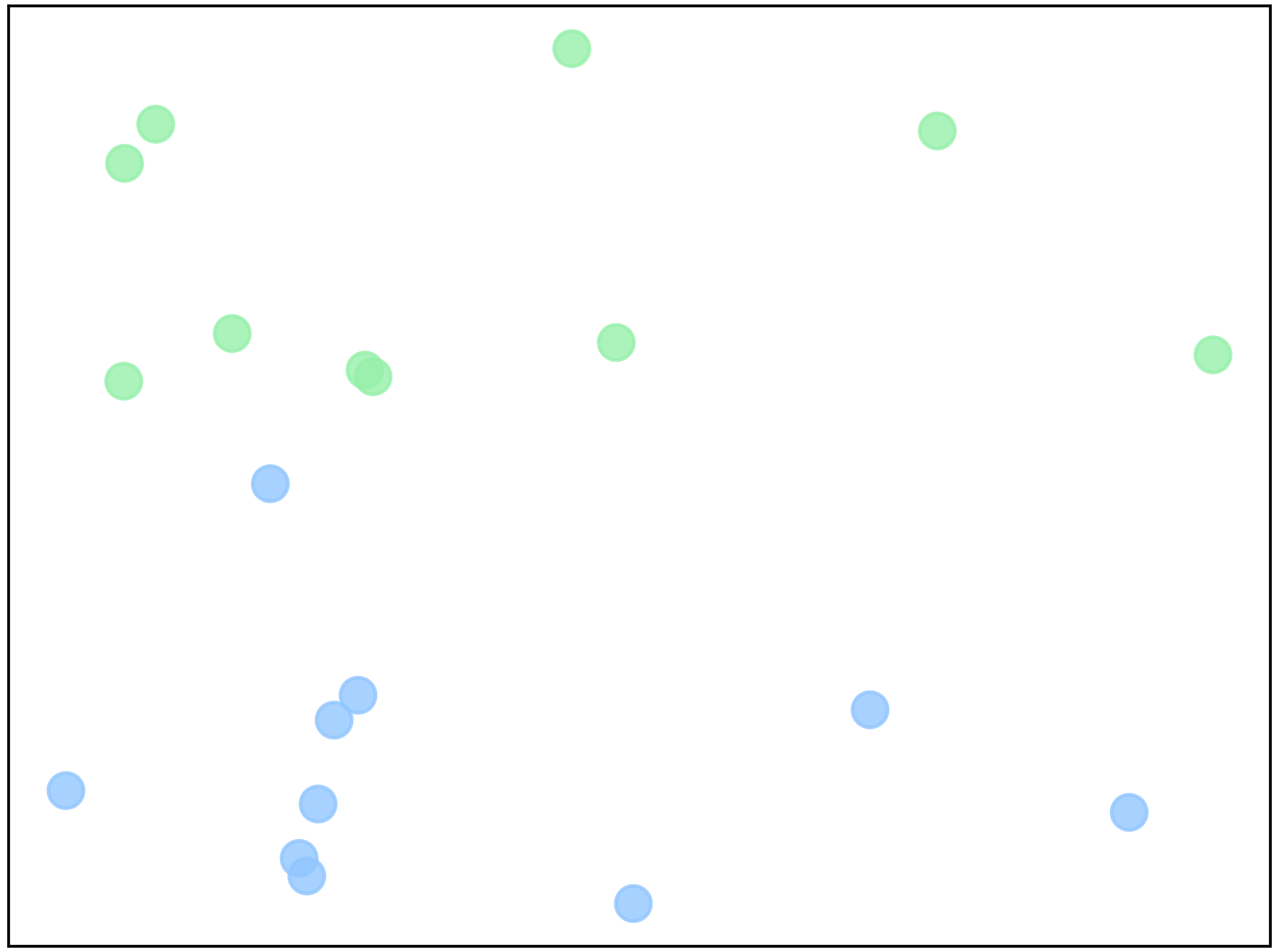
Low Dimensional

$$\mathcal{D}_{data} \approx \mathcal{D}_{train} \approx \mathcal{D}_{valid} \approx \mathcal{D}_{test}$$



High Dimensional

$$\mathcal{D}_{data} \neq \mathcal{D}_{train} \neq \mathcal{D}_{valid} \neq \mathcal{D}_{test}$$



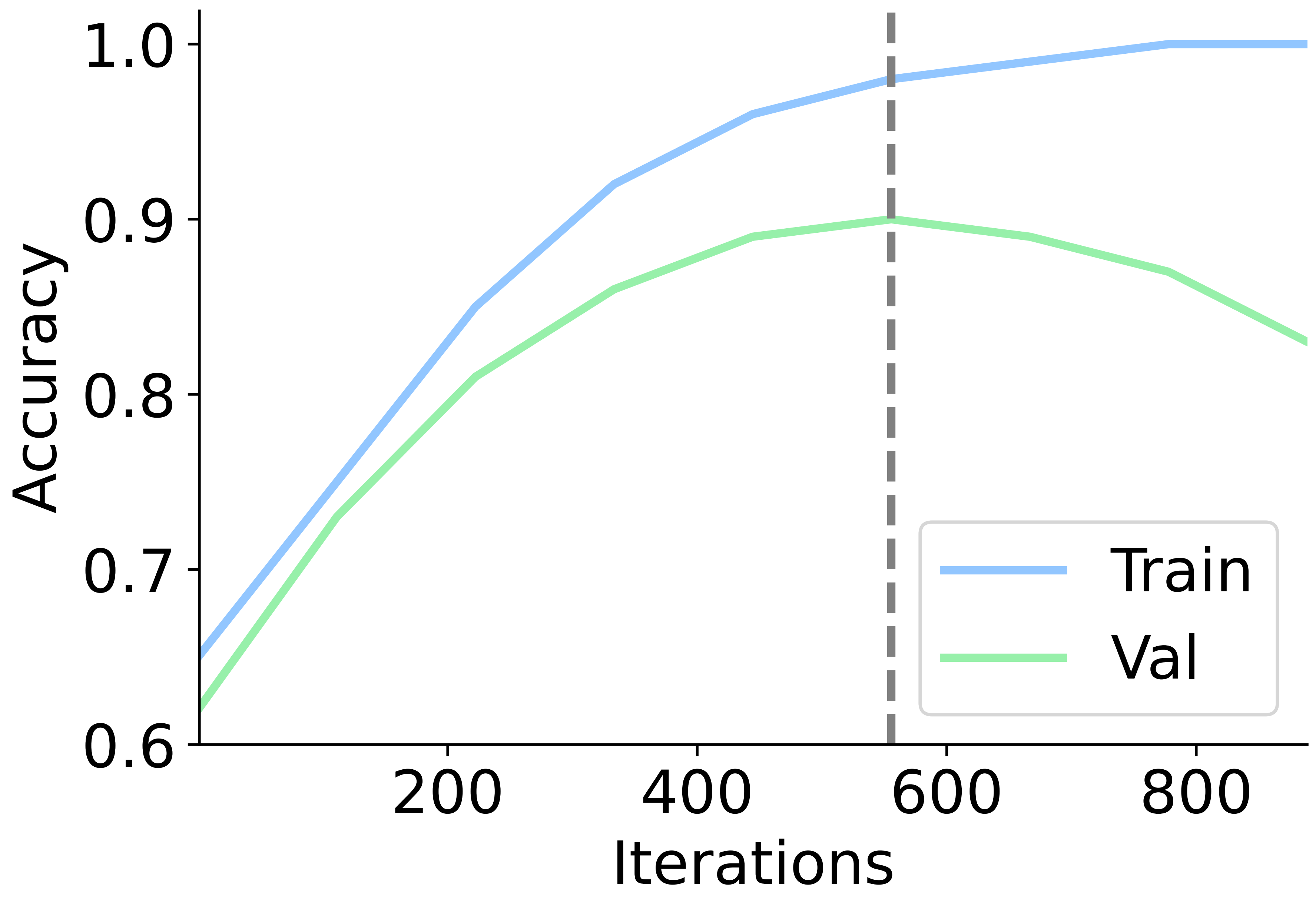
Can we overfit with infinite training data?

No

- Never train on the same data instance

Preventing Overfitting: Early Stopping

Stop Training When Validation Accuracy Peaks

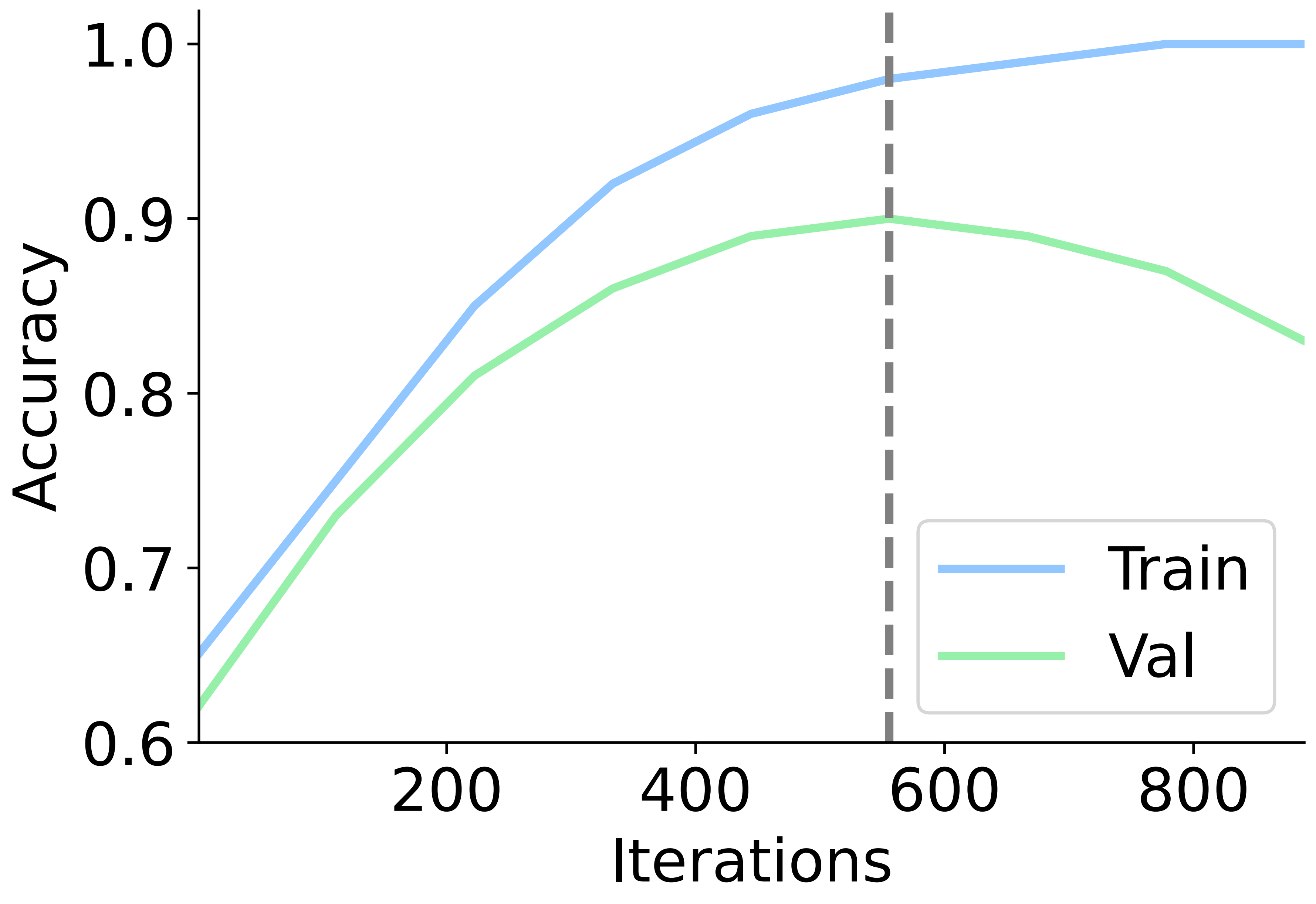


Early Stopping in Practice

No need for manual stop button

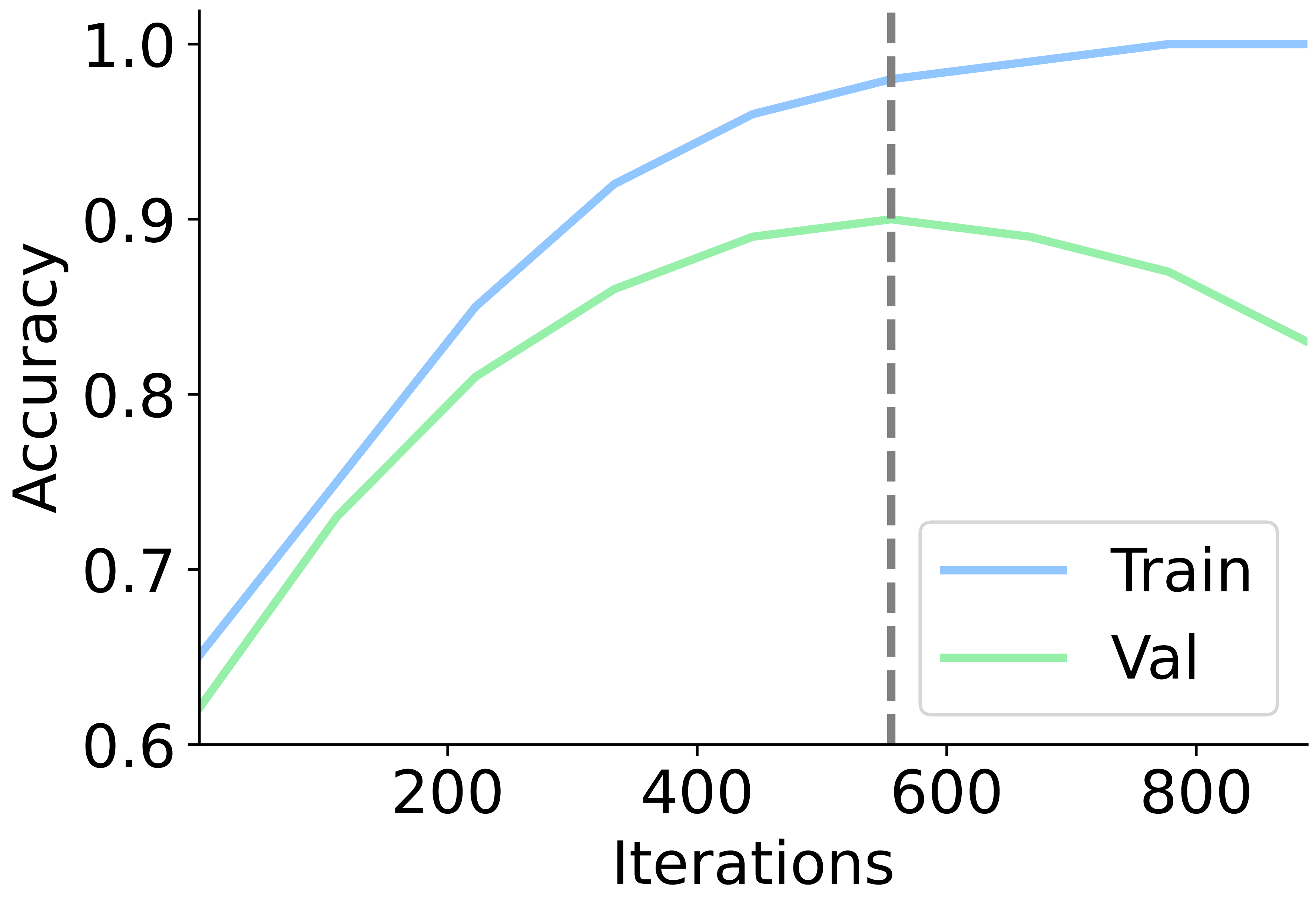
Every few epochs

- Measure validation accuracy
- Save your model

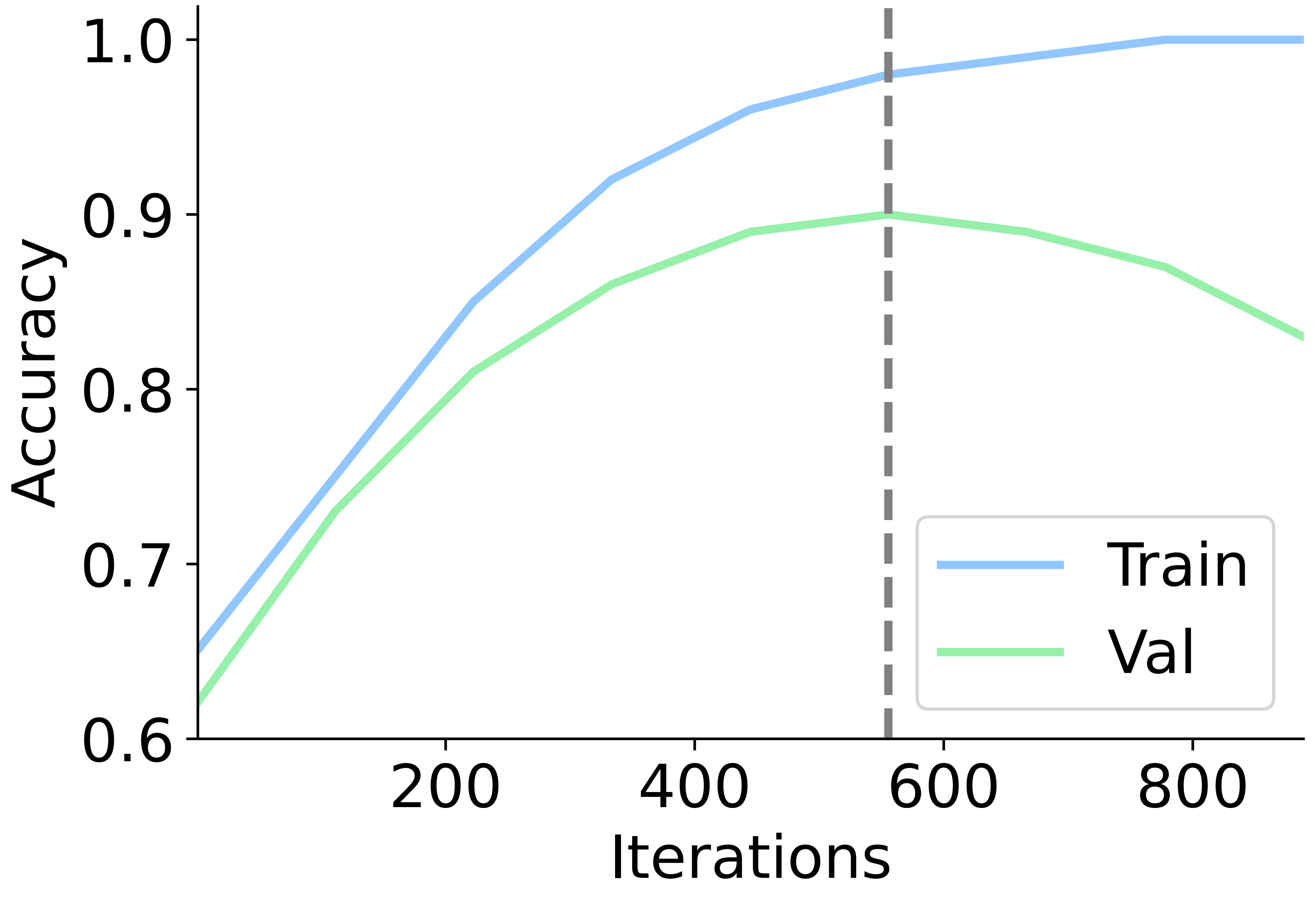


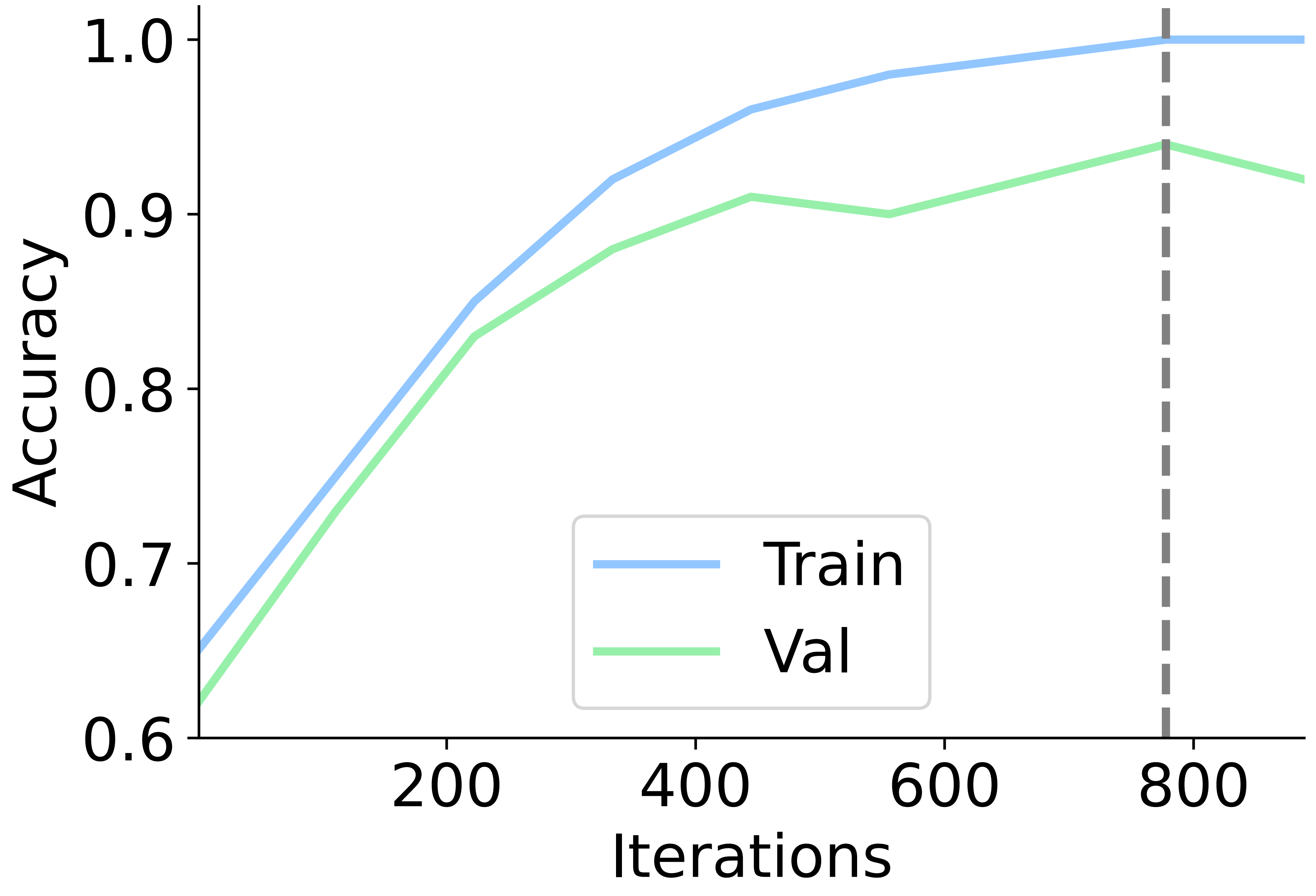
When Do We Overfit?

When we train on the same data multiple times



More Data Delays Overfitting



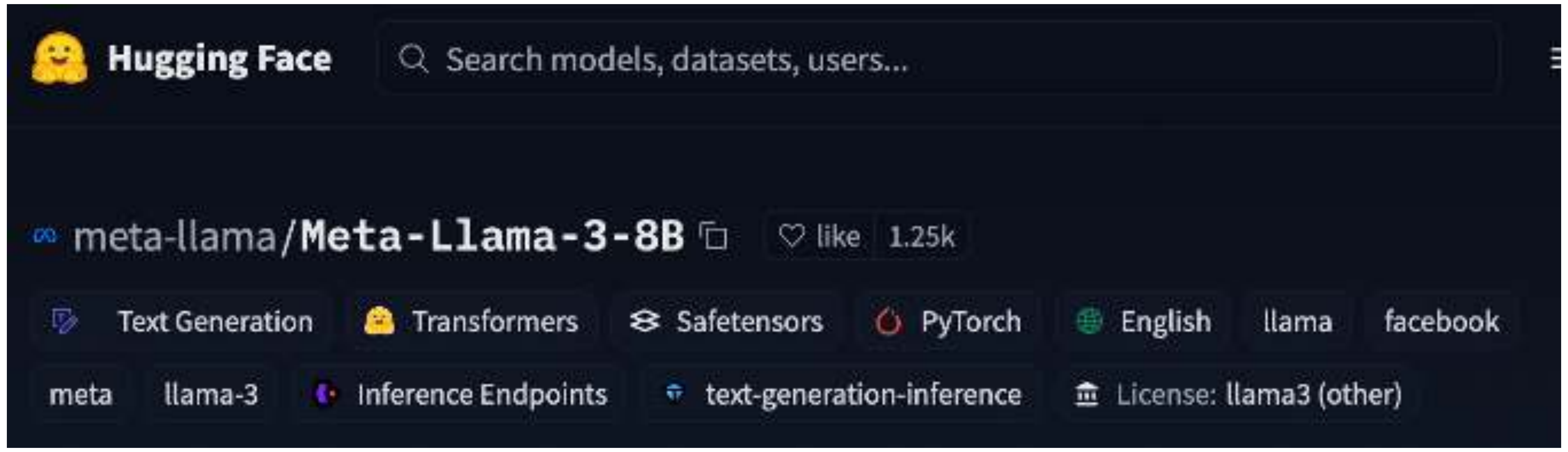


Practical Example: Large Language Models

Huge corpus of training data from the internet

Never sees the same data twice during training

Cannot overfit if < 1 epoch



How can I help you today?

What if We Cannot Get More Data?

Data Augmentation

- Make more data from our existing data
- Randomly transform data during training
- Reuse/Rephrase labels

"pink primose"

"pink primose"

"pink primose"

"pink primose"

Preventing Overfitting: Image Augmentations

Original

Tint/hue

Brightness

Crop

Rotate

Scale

Saturation

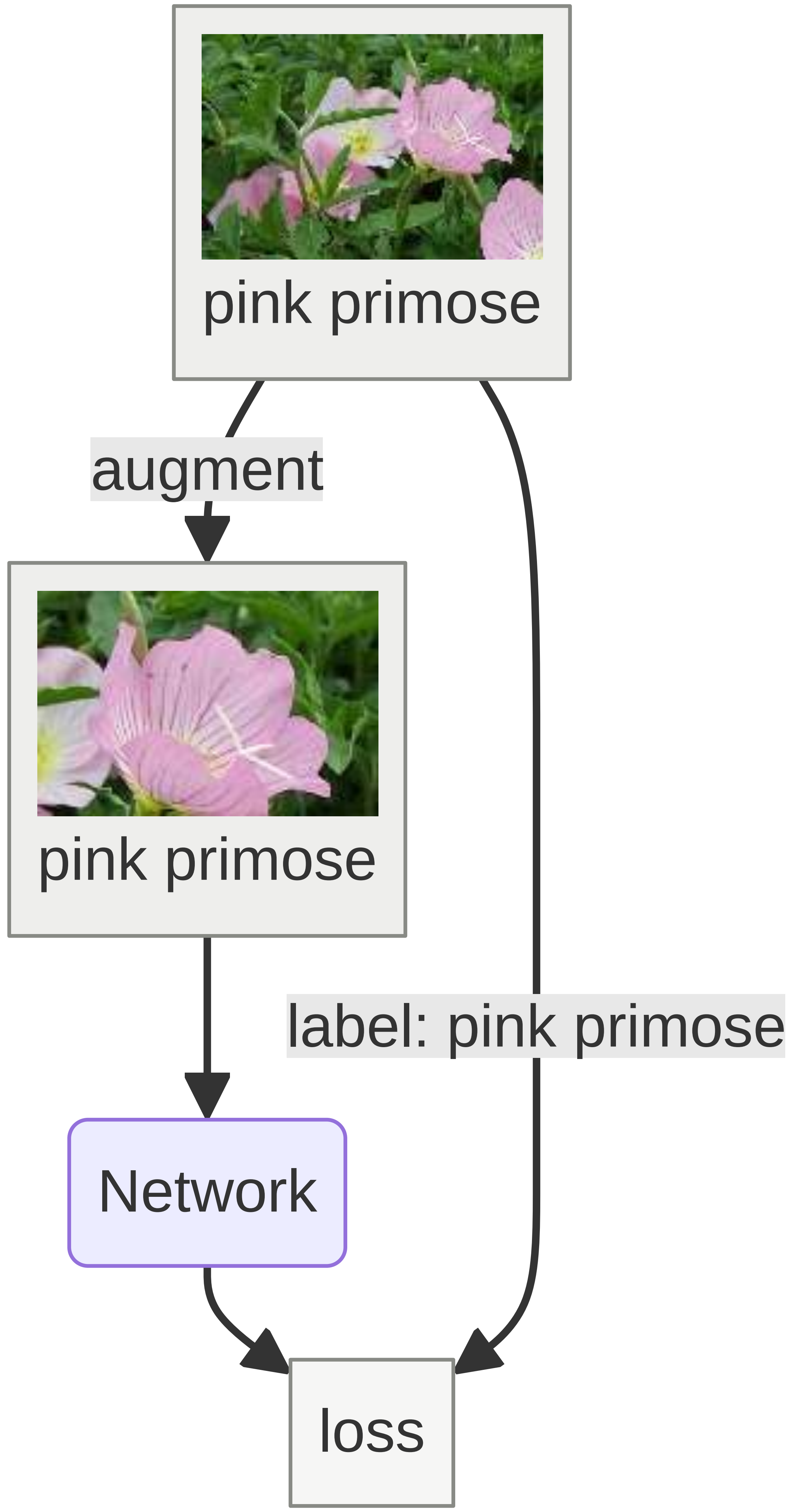
Grey

Flip

Training With Data Augmentation

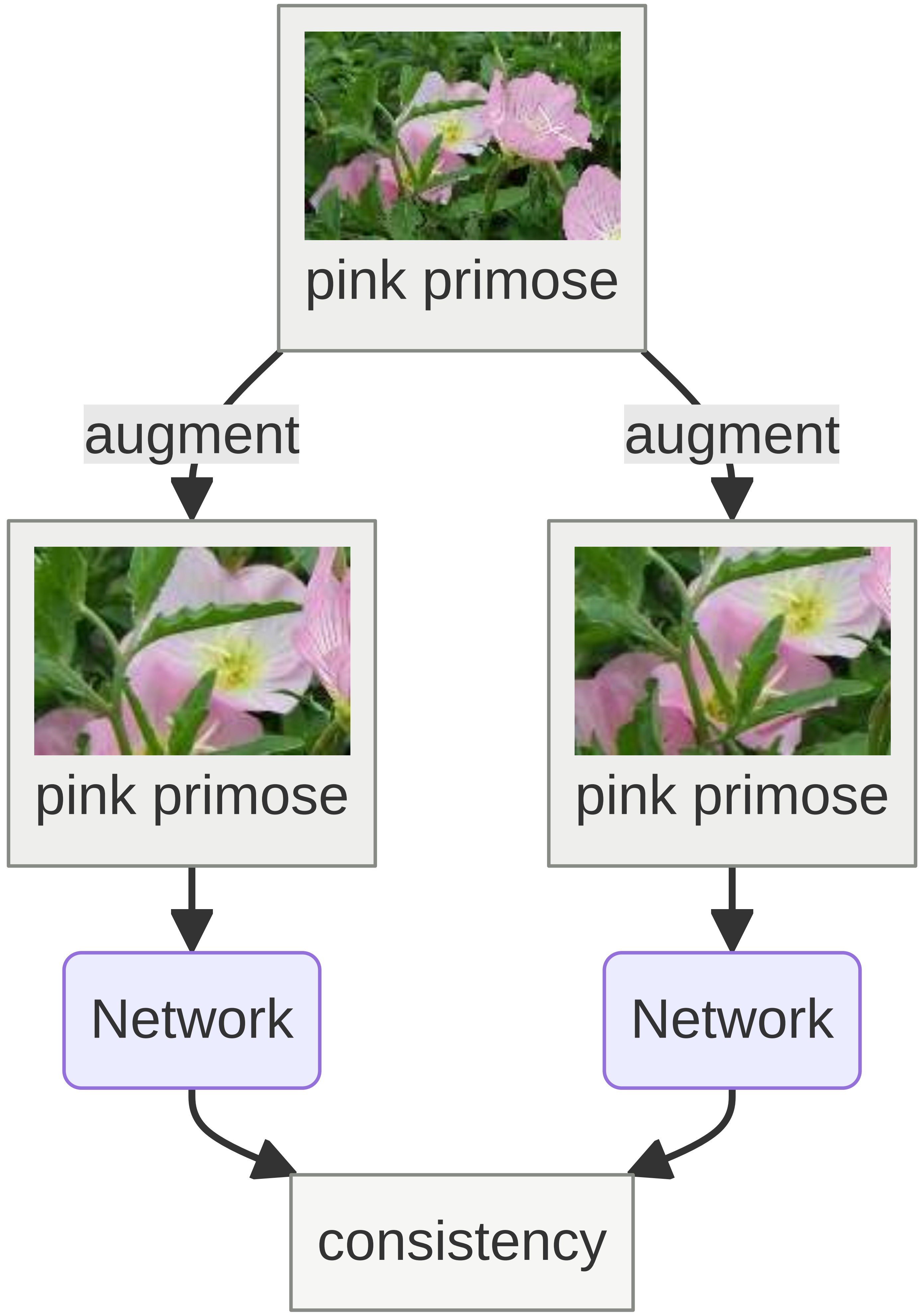
Randomly augment every single iteration

Network never sees exact same data twice



Unsupervised Data Augmentation

Captures invariances on unseen and unlabeled data[1]:

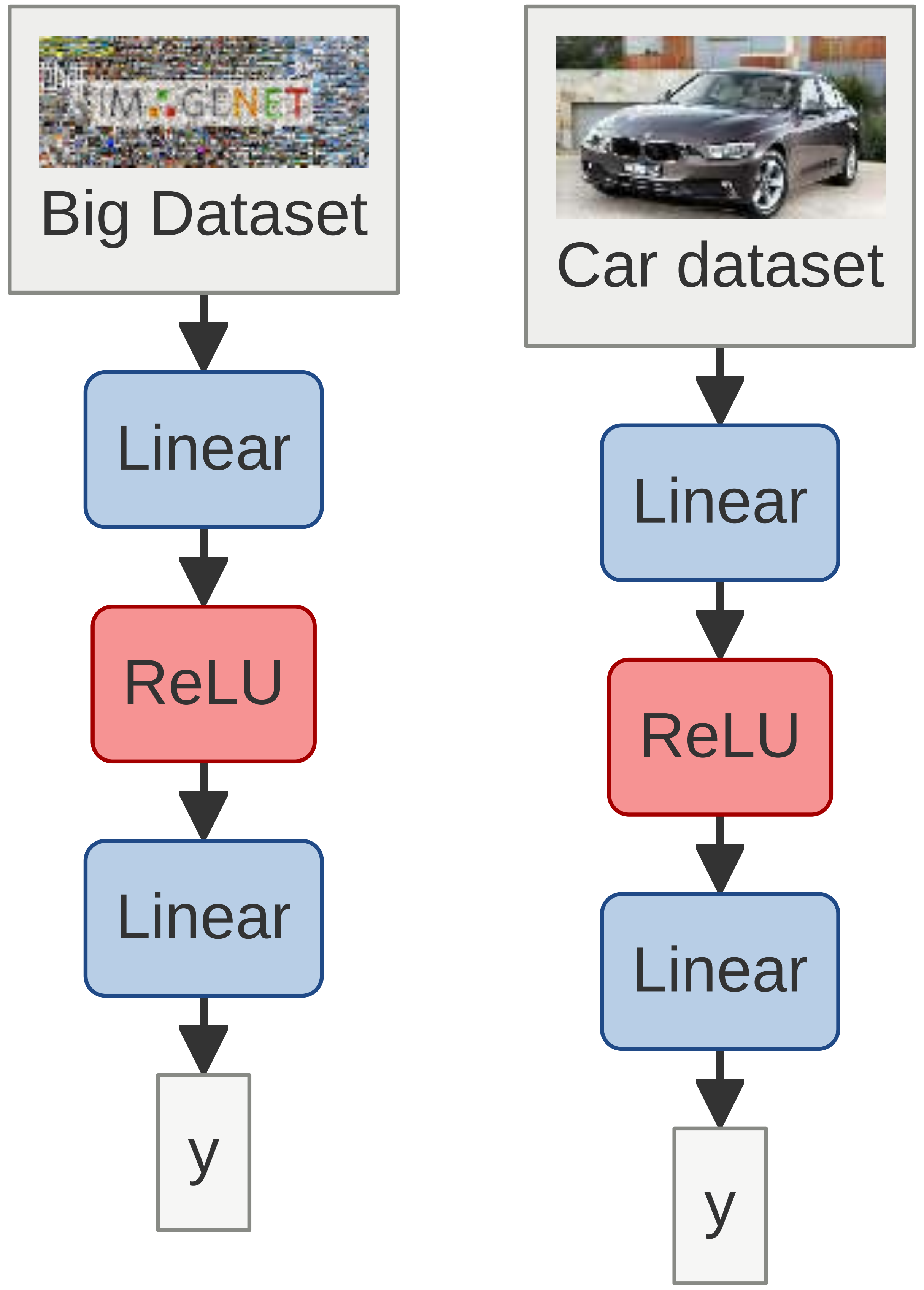


1. Xie, Dai, Hovy, Luong, Le, "Unsupervised Data Augmentation for Consistency Training", NeurIPS 2020 ↩️

What if we still don't have enough data?

Transfer Learning

- Train model on large dataset (pre-training)
- Continue training on target dataset (fine-tuning)



Preventing Overfitting: Pre-Training

Computer vision

- Supervised (e.g. ImageNet)
- Self-supervised (e.g. MAE)

Natural Language Processing

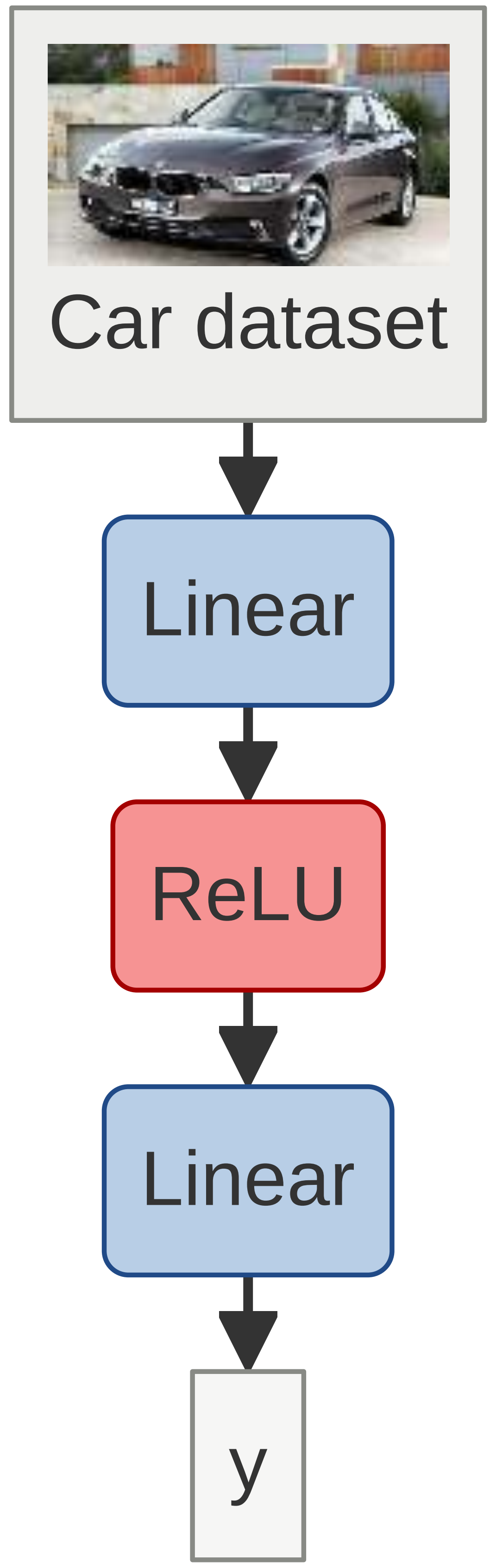
- Self-supervised (e.g. Wikipedia)

WIKIPEDIA
The Free Encyclopedia

Pre-training / fine-tuning in practice

Download a pre-trained model

Run a few training iterations on small dataset



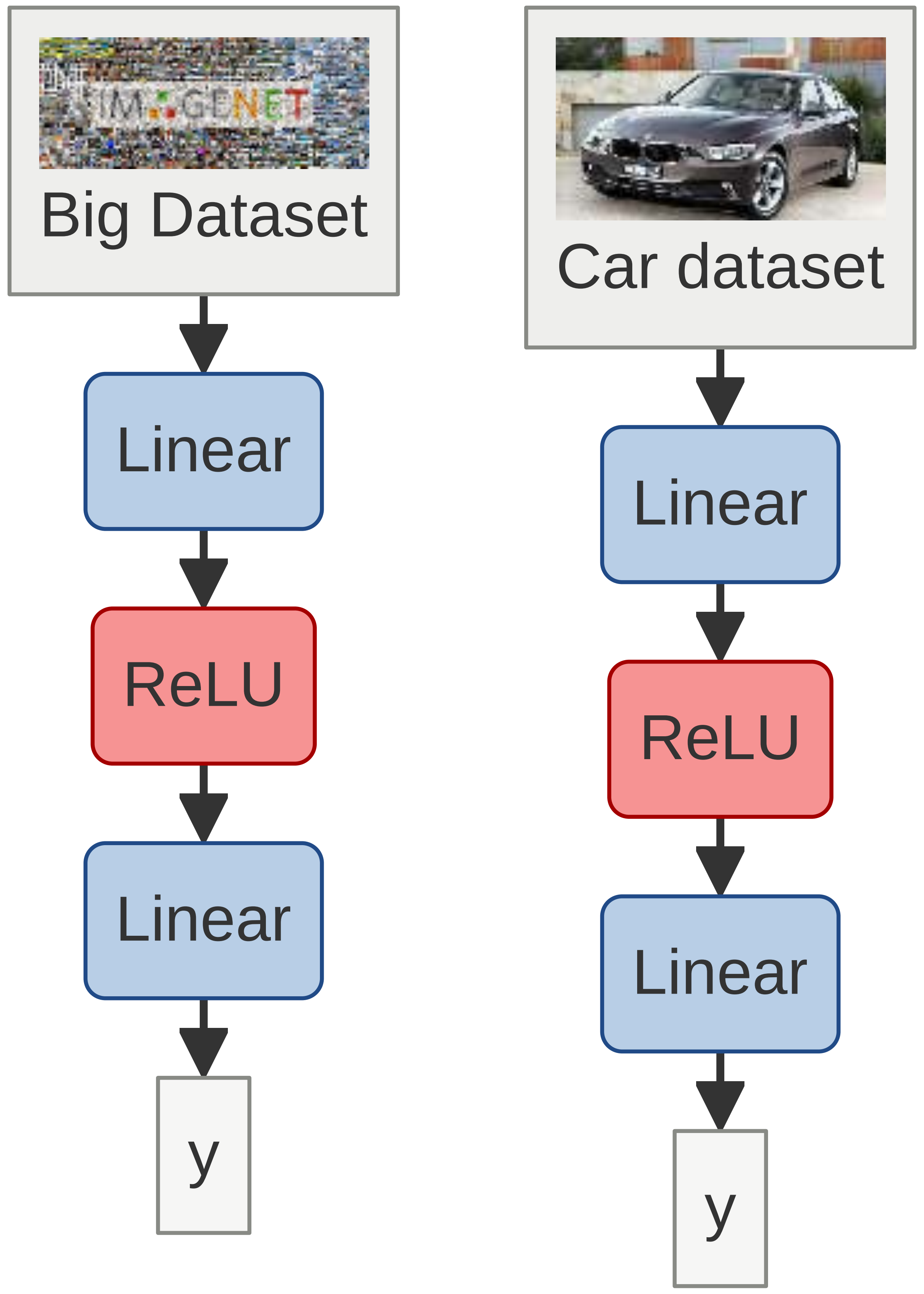
Why Does Transfer Learning Work?

Similar inputs

- e.g. images, text, …
- Transfer between tasks

Good initialization

- Learned weights are initialized well
- Better init allows for better training



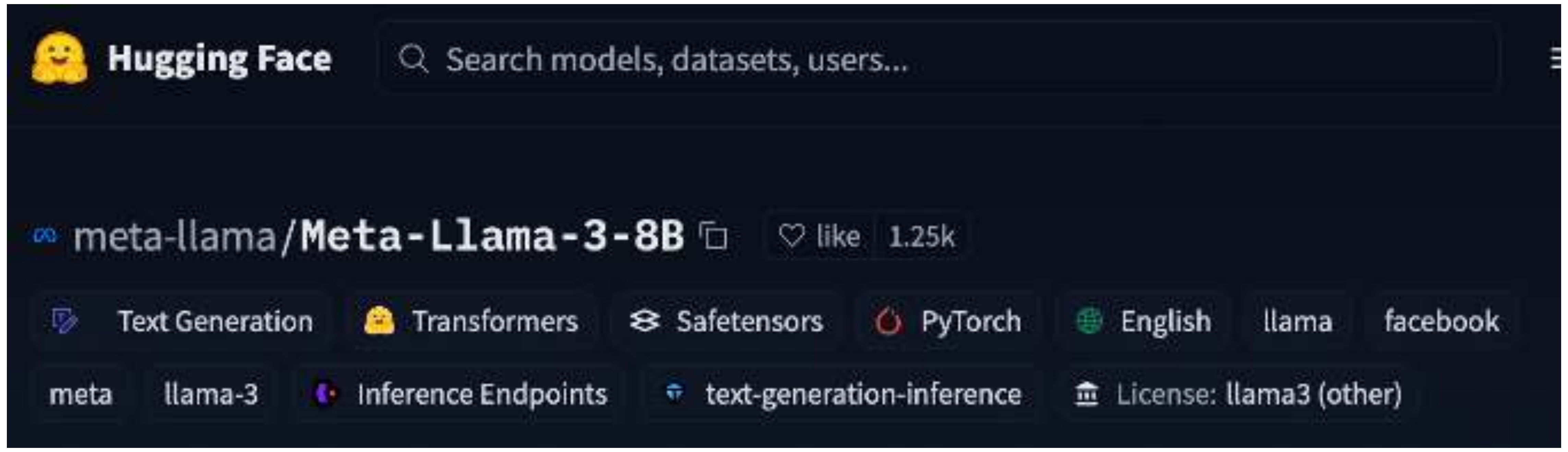
When to Use Transfer Learning?

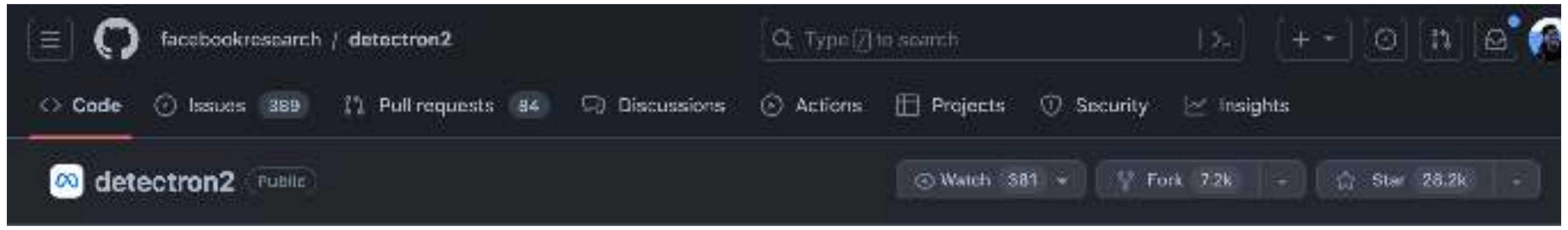
Whenever possible!

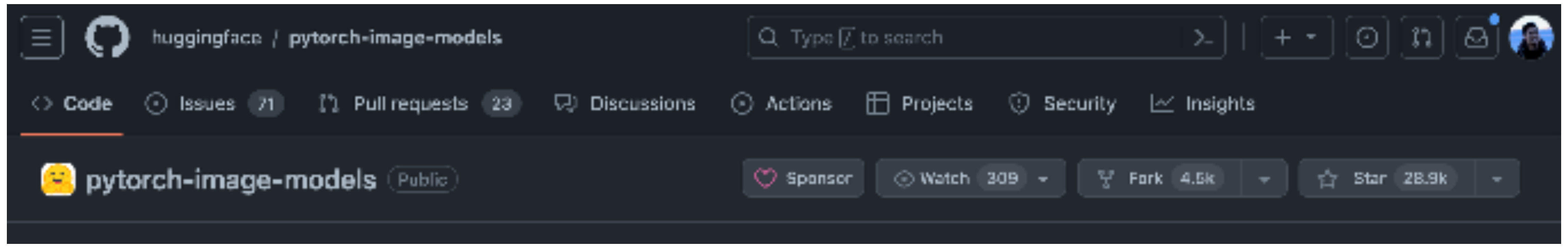
- In early experiments
- Large pre-trained model exists

Where can we find models?

- Github
- Huggingface
- Detectron2
- ...





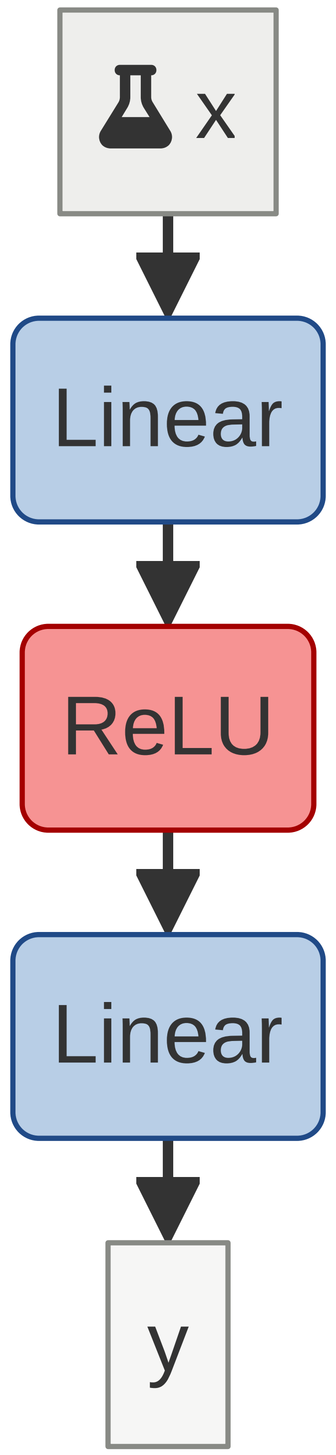


Why Does Our Model Overfit? - Part I

Model exploit patterns that exist in training data

These patterns are not in the validation / test data

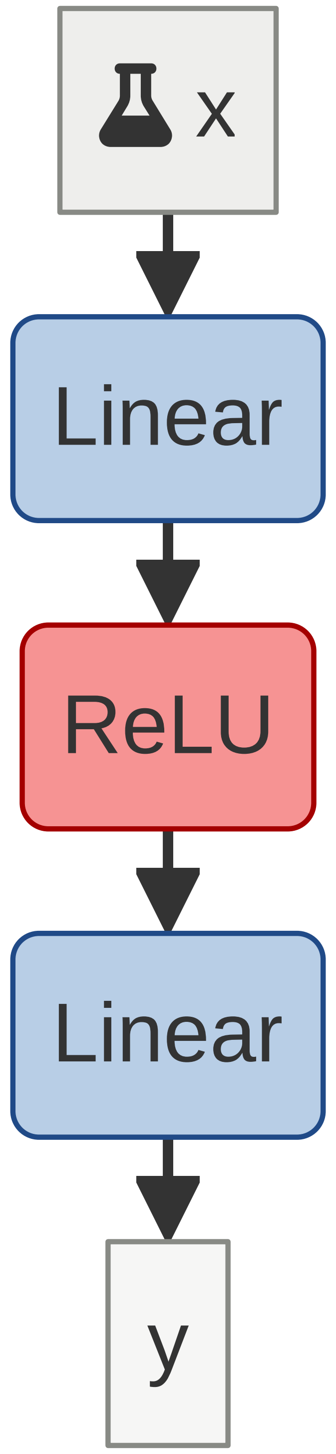
Not all activations overfit



Why Does Our Model Overfit? - Part I

Deeper layers overfit more

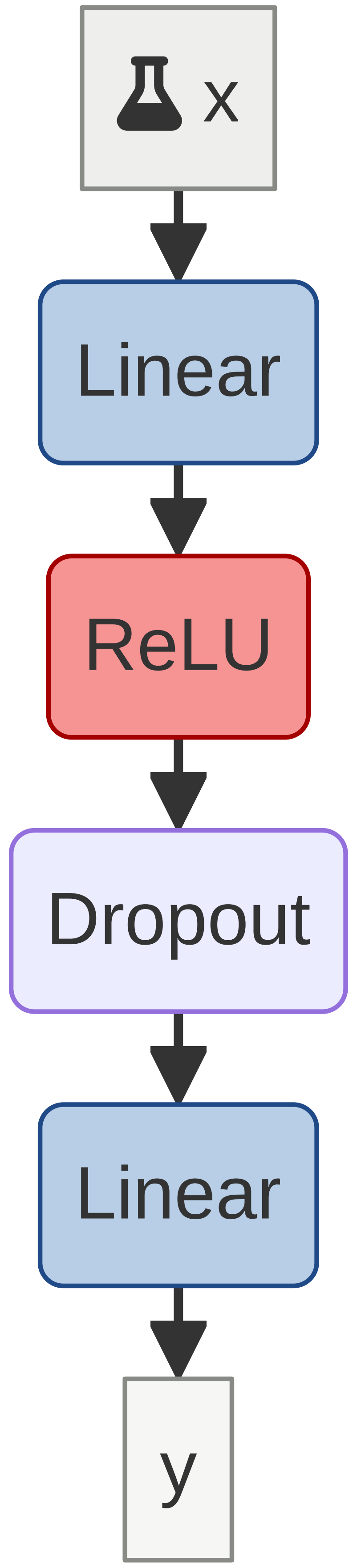
Relies on overfit activations from previous layers



Preventing Overfitting: Dropout

Method: Randomly remove activations

Reduces reliance on specific activations in previous layer



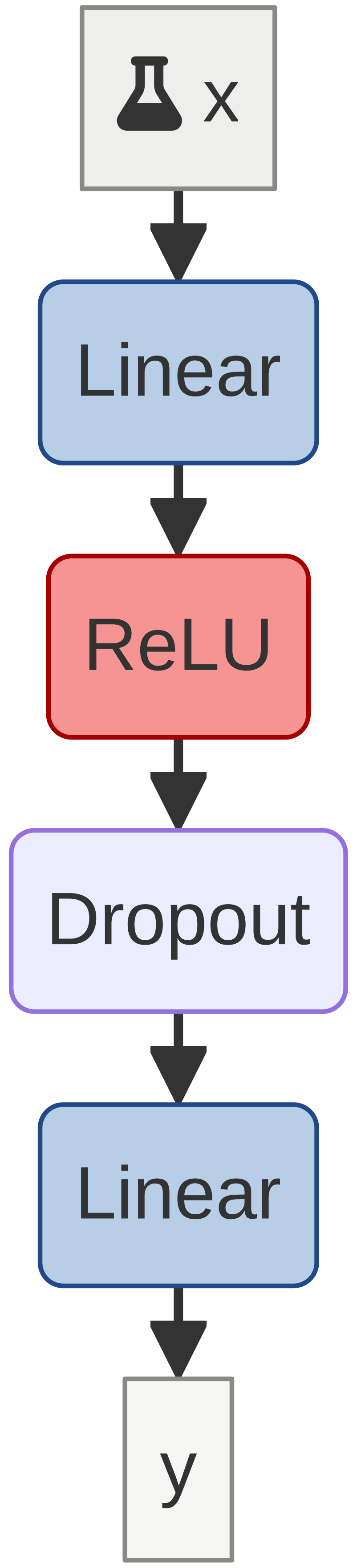
Preventing Overfitting: Dropout

During training

- With probability α set activation $a_l(i)$ to zero

During evaluation

- Use all activations but scale by $1 - \alpha$



Dropout in Practice

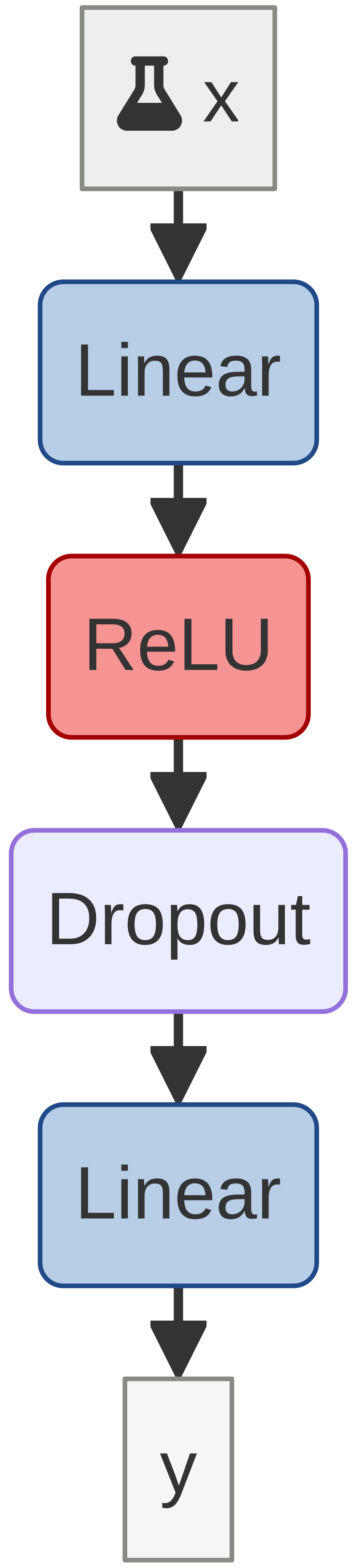
A separate "layer" `torch.nn.Dropout`

During training

- With probability α set activation $a_l(i)$ to zero
- Scale activations by $\frac{1}{1-\alpha}$

During evaluation

- Identity
- **Important:** do not forget to call `model.eval()`!

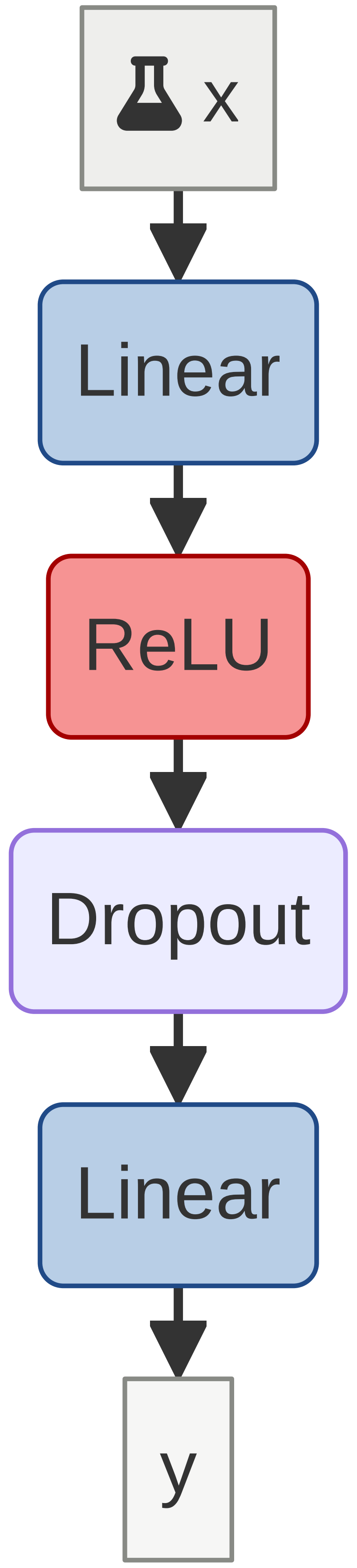


Where to Add Dropout?

Before any large fully connected layer

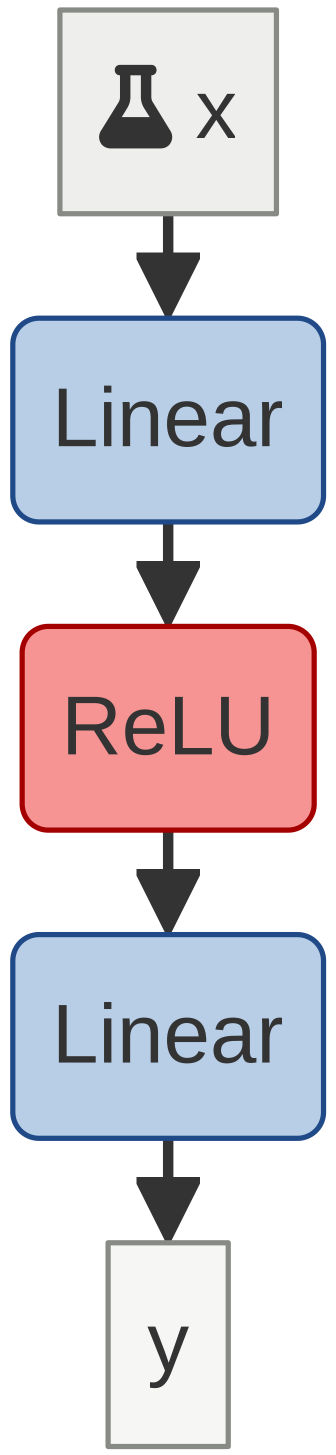
Before some 1x1 convolutions

Not before general convolutions



Why Does Our Model Overfit? - Part II

Models becomes too complex and large

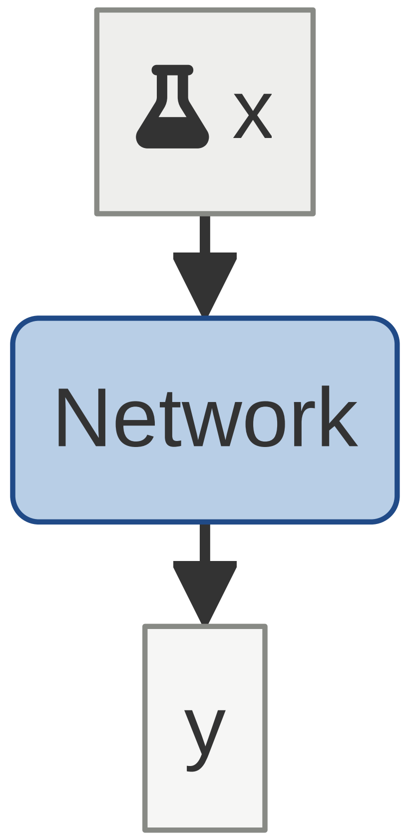


Idea 1: Smaller Model

✓ Smaller models overfit less

✗ Smaller models fit worse

✗ Smaller models generalize worse



Idea 2: Big Model With Regularization

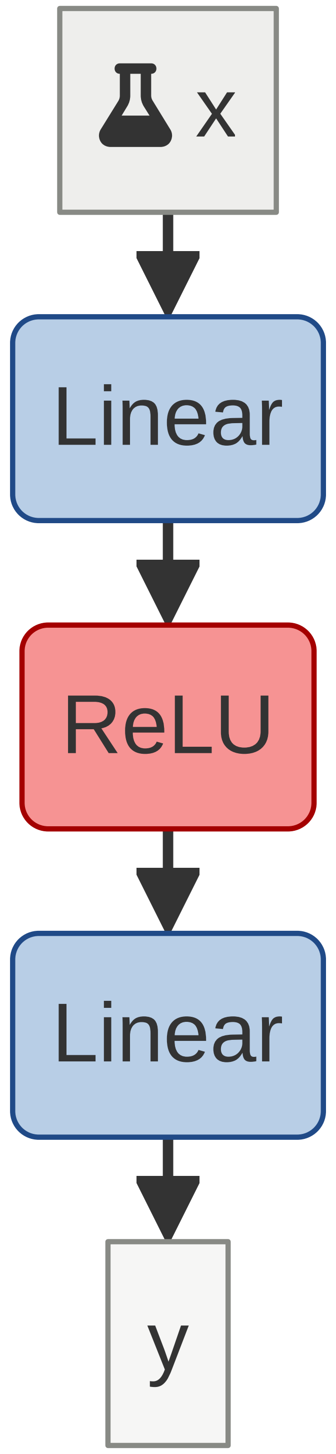
Weight Decay

✓ Keep weights small (L2 norm)

✓ Keep weight at same magnitude

Other reasons to use weight decay

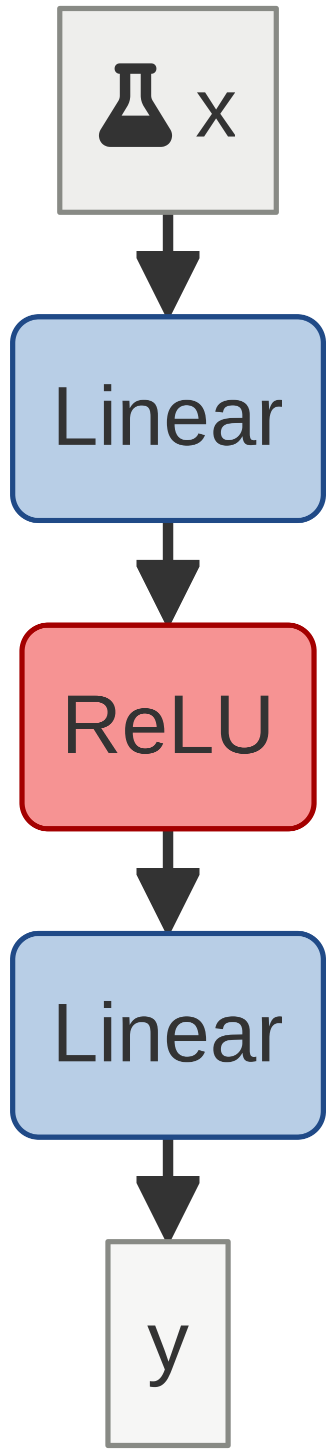
✓ Helps with exploding gradients



Idea 2: Big Model With Regularization

AdamW

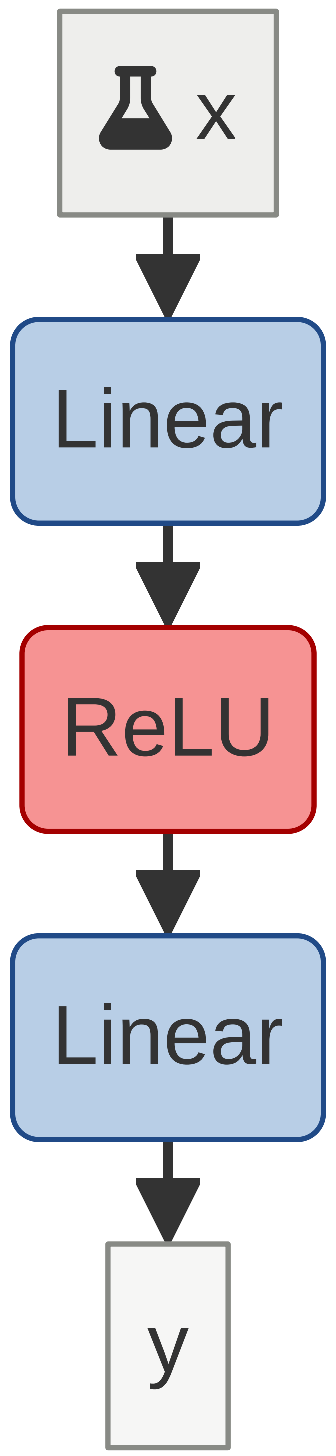
```
m, v, t = 0, 0, 1
for epoch in range(n):
  for (x, y) in dataset:
    J = ∇l(θ|x,y)
    m = (1-β_1) * J + β_1 * m
    v = β_2 * v + (1-β_2) * J.square()
    m = m / (1 - β_1^t)
    v = v / (1 - β_2^t)
    b = m / v.sqrt()
    θ = θ - ϵ * (b.mT + decay * θ)
    t += 1
```



How to Use Weight Decay?

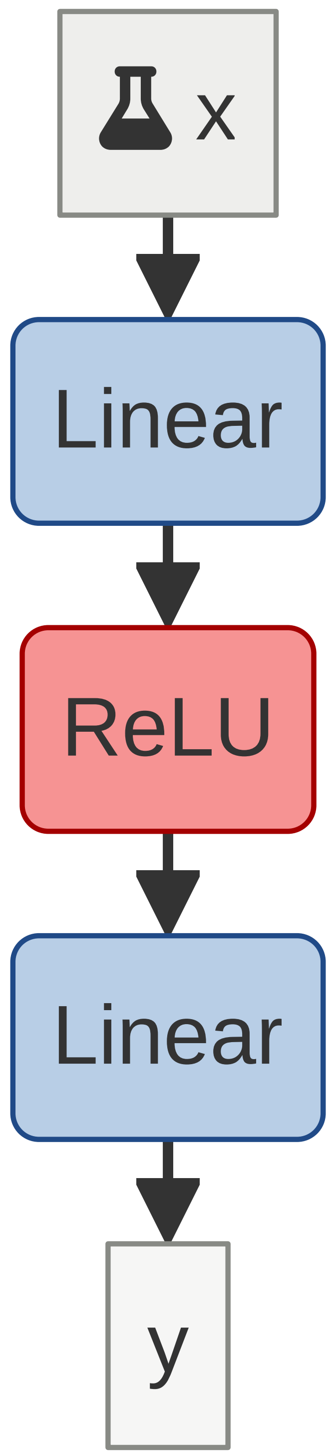
Parameter in optimizer

```
torch.optim.AdamW(lr=lr, weight_decay=1e-4)
torch.optim.SGD(lr=lr, weight_decay=1e-4)
```



Why Does Our Model Overfit? - Part III

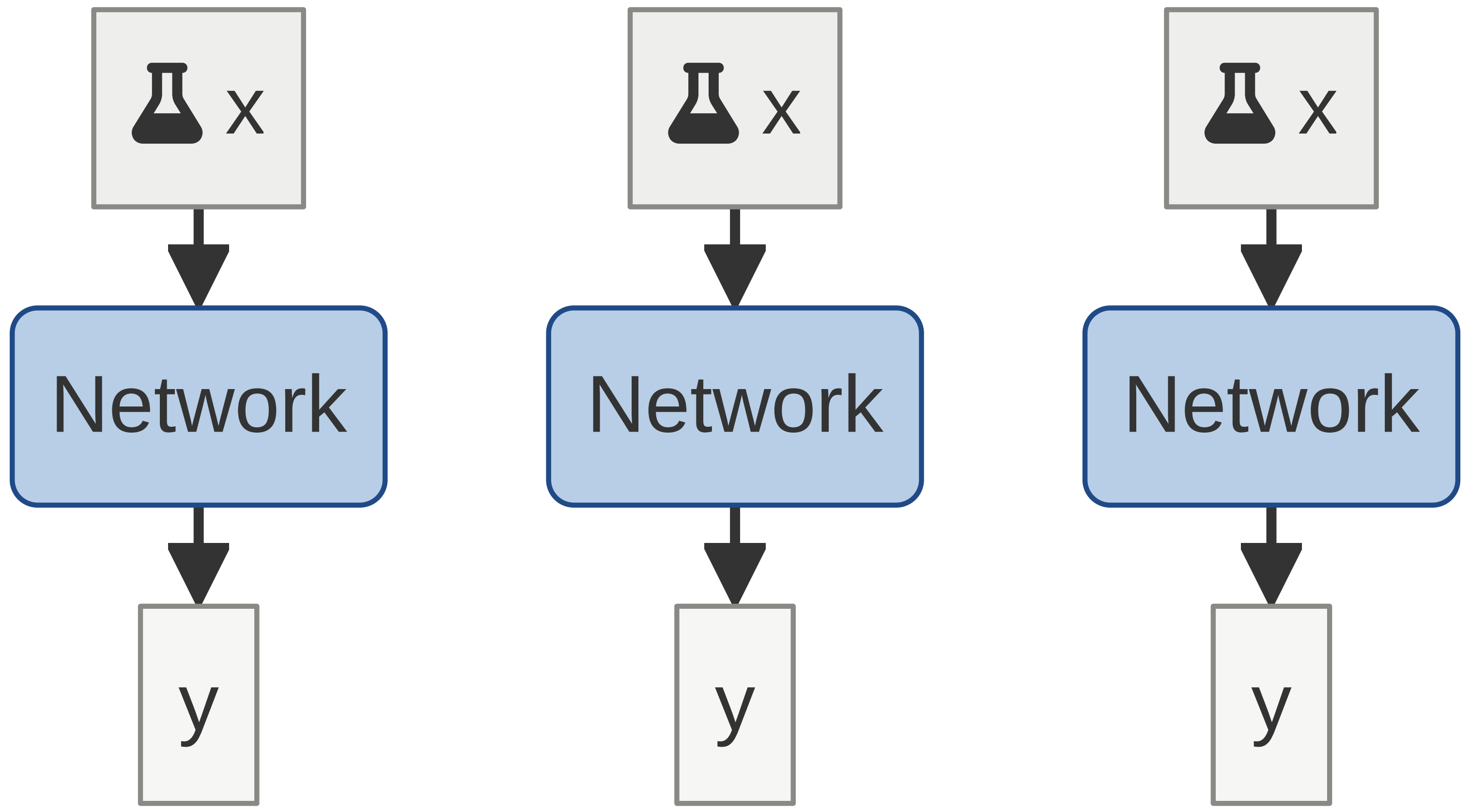
Models are too complex



Preventing Overfitting: Ensembles

Train multiple small models

Average predictions of multiple models



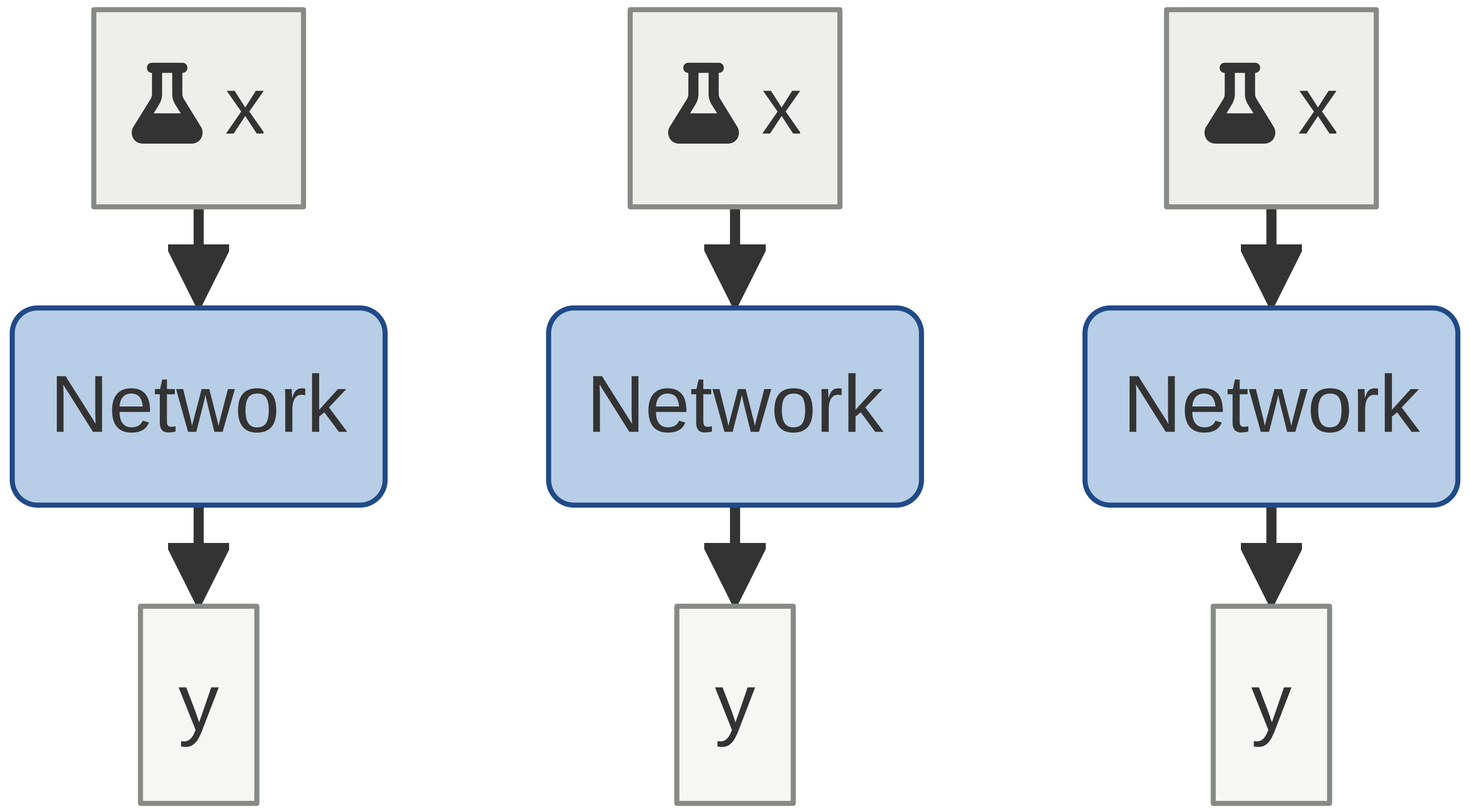
Preventing Overfitting: Ensembles

Pre-deep learning

- Use different subsets of training data

Deep learning

- Use different init / data augmentations
- Different local minima



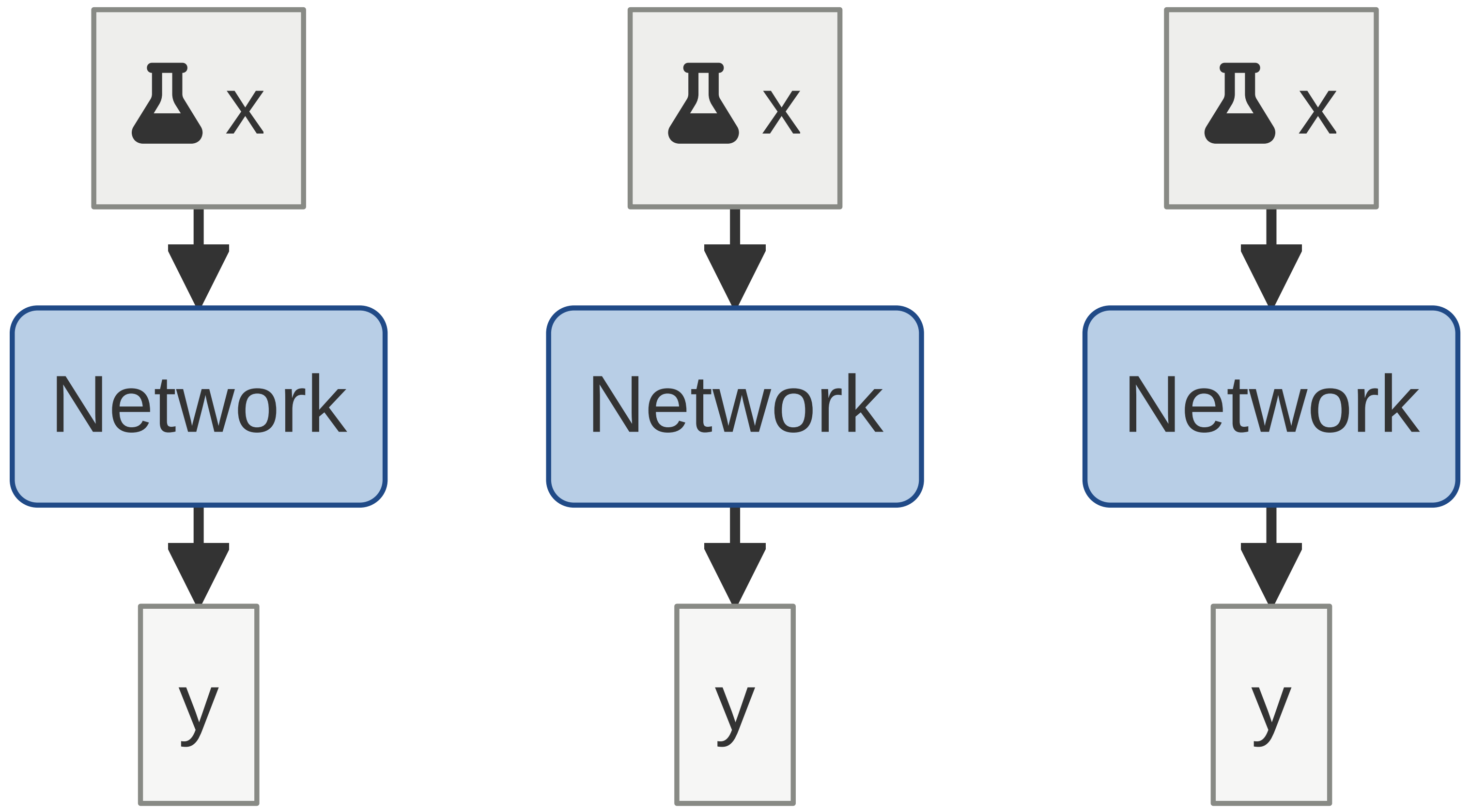
Why Do Ensembles Work?

Fewer parameters / model

Each model overfits in its own way

Usually a 1-3% accuracy boost on most tasks

- longer training

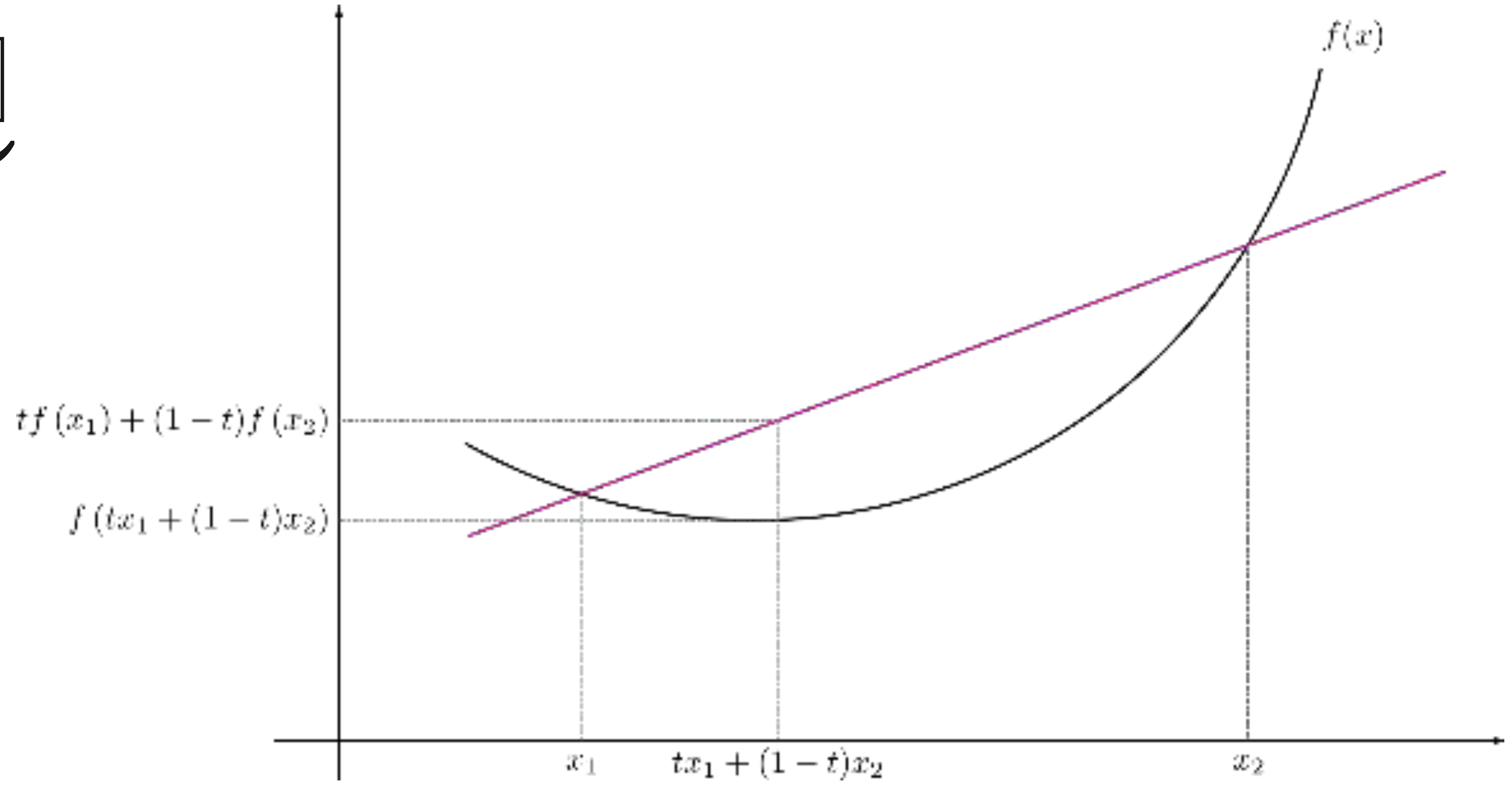


Why Do We Average Predictions?

$$\underbrace{\mathbb{E}_{(\mathbf{x},\mathbf{y})}[l(\frac{1}{M}\sum_{i=1}^{M} f(\mathbf{x}|\theta_i), \mathbf{y})]}_{\text{loss for ensemble}} \leq \frac{1}{M}\sum_{i=1}^{M} \underbrace{\mathbb{E}_{(\mathbf{x},\mathbf{y})}[l(f(\mathbf{x}|\theta_i), \mathbf{y})]}_{\text{loss for model } i}$$

for a convex loss function l and M models

follows from Jenson’s inequality [1]:



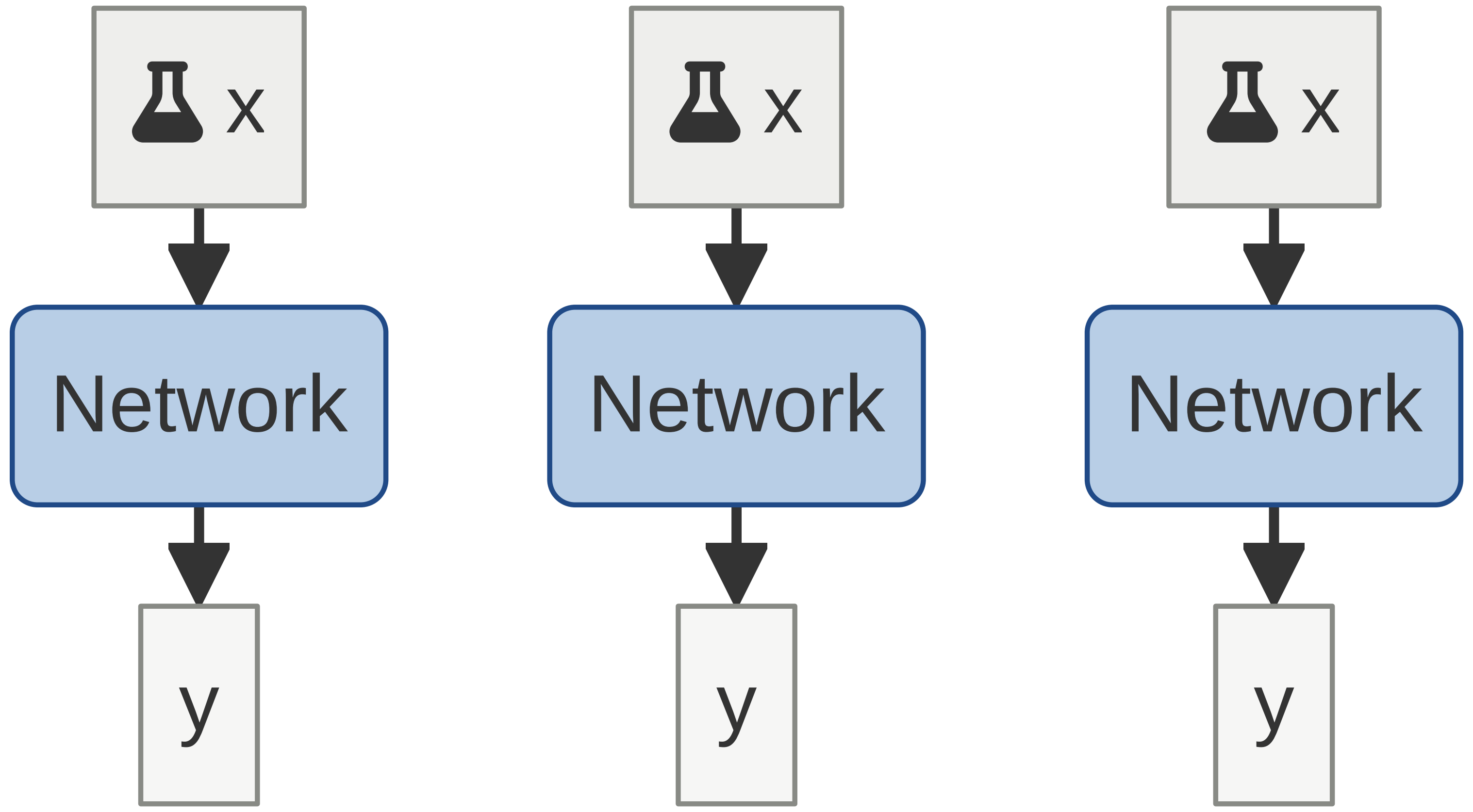
1. https://en.wikipedia.org/wiki/Jensen's_inequality ↩

When to Use Ensembles?

If you have the compute power

If you really need the last bit of accuracy

- e.g. production, competitions



(Language) Transformers in PyTorch