A Study on the Effects of Representation Bias on AI Performance and Methods of Mitigating it

Sarah Abdelazim
Morris Chan
Julie Song

Mentor: Simon Goring

Introduction & Objectives

Data Science Techniques

Data Product

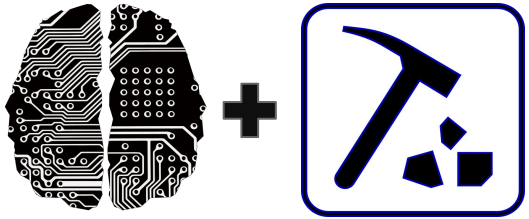
Results

Conclusions and Recommendations

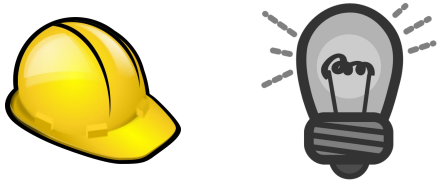
GOLDSPOT DISCOVERIES CORP.

"Mineral exploration is not a new business. But with a **stalled rate of discoveries**, the global supply chain is faced with **dwindling supply and surging demand**. With fewer, smaller and more expensive discoveries, the industry is finally doing things differently."

– https://goldspot.ca/



Combines AI with mineral exploration projects



Reduce risk, increase efficiency

Unlock full potential of historical data

The Dataset

Class 2

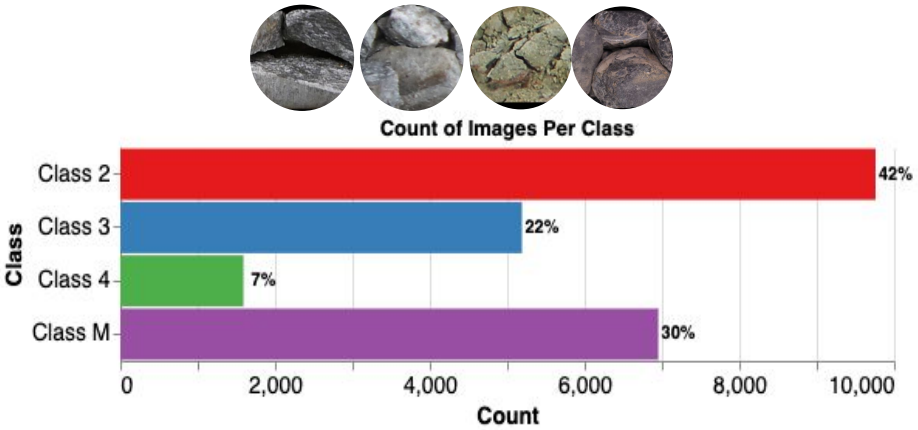
Class 3

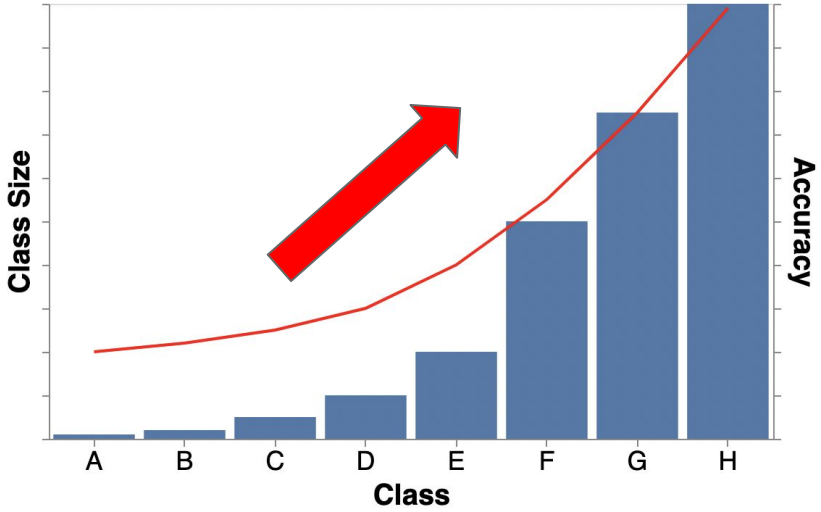
Class 4

Class M

Mining core box (ref)

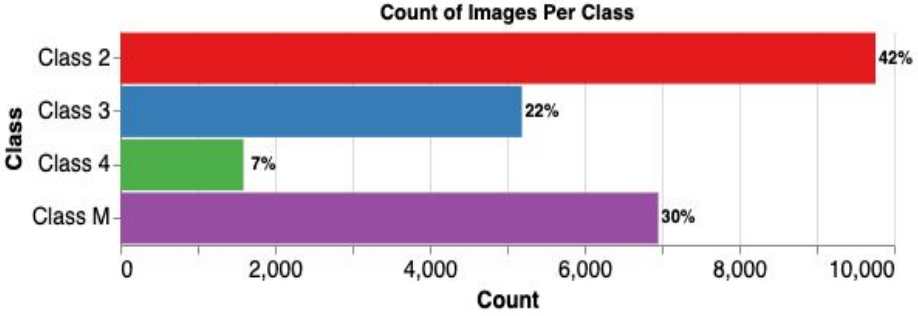
Data Imbalance





“How can we make accurate predictions for all 4 classes?”

Data Imbalance



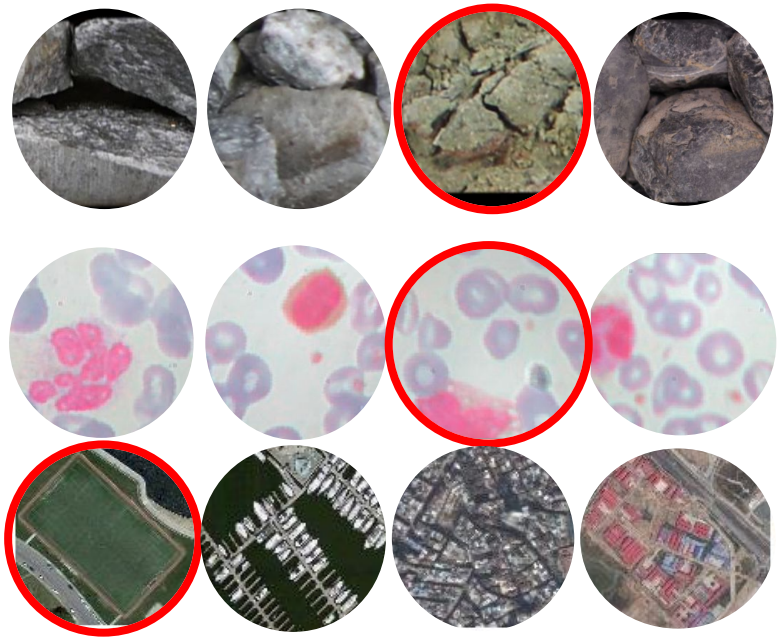
"How can we make accurate predictions for all 4 classes?"

Predicted Class

Actual Class	Safe	Risky
Safe	Good!	Opportunity Costs of New Projects
Risky	Accidents and Injuries	Good!

The General Problem

Data imbalance is a very general problem.

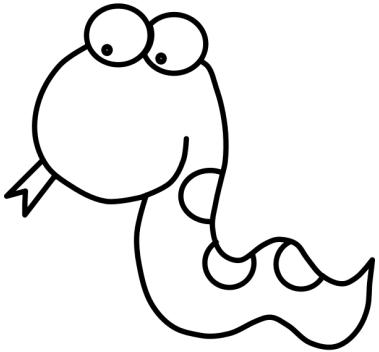


Potential Solutions:

- ★ Data preprocessing
- ★ Model selection
- ★ Loss function implementation
- ★ Optimizer choice

Scientific Objectives

```
import debiaser
```



★ A package that is able to train and compare models, and use different techniques based on the potential solutions to tackle data imbalance

★ Success Criteria:
- Interpretable results
- Improved performance on imbalance dataset

Data Science Techniques

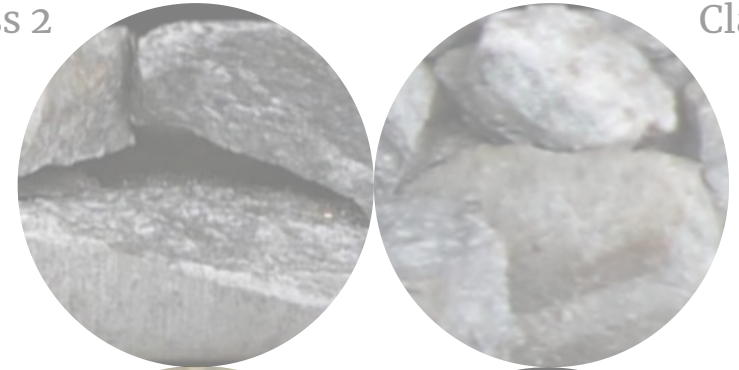
Data Product

Results

Conclusions and Recommendations

The Dataset

Class 2

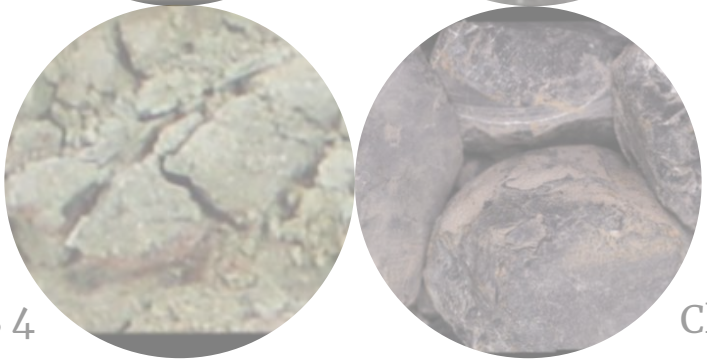


Class 3

Lotus

Orchid

Class 4

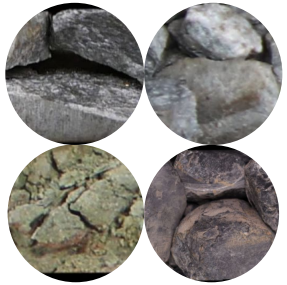


Class M

Tulip

Sunflower

Debiasing Methods Overview



Input

Data

Transformation

Pre-trained Model

Criterion

Optimization Algo.

Training & Evaluation

Visualization

Resampling Methods

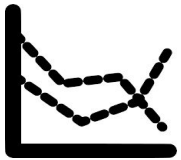
torchvision.transforms

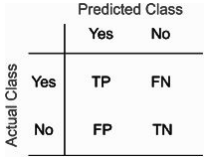
torchvision.models

Loss Functions

Parameter Regularization

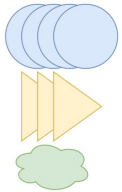
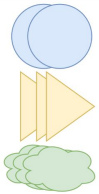
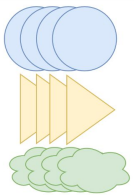
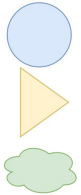
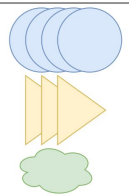
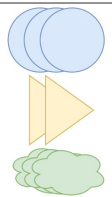
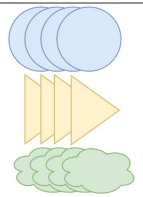
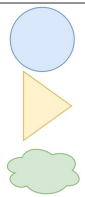
Output

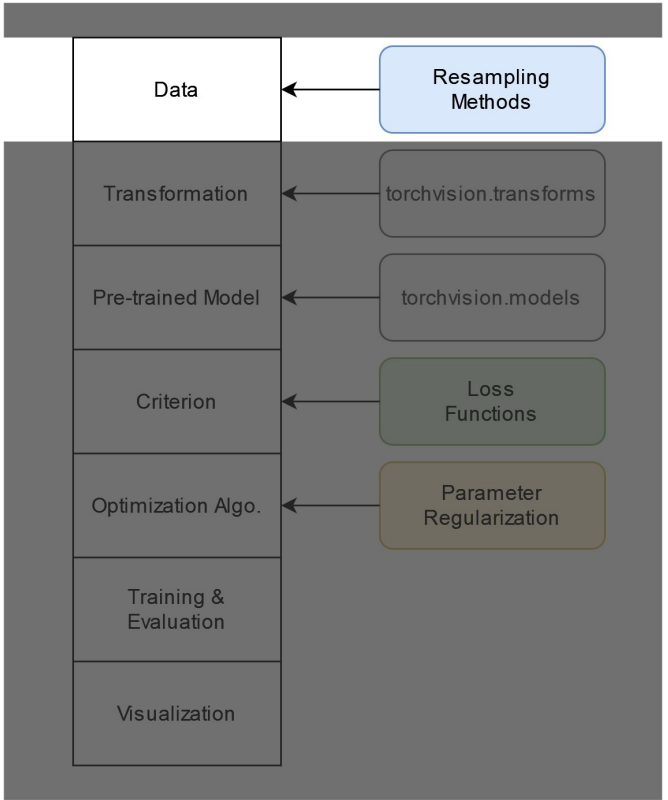




Resampling

```
from debiaser.general import load_image

load_image(
      ...,
      sampling_strategy='weighted')
```

Epoch	Original	Weight Random Sampling	Oversampling	Undersampling
1				
2				

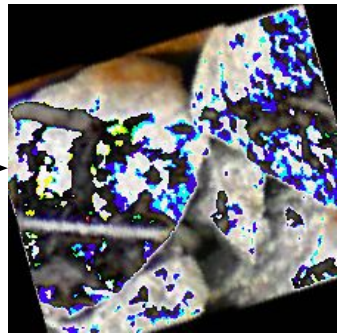


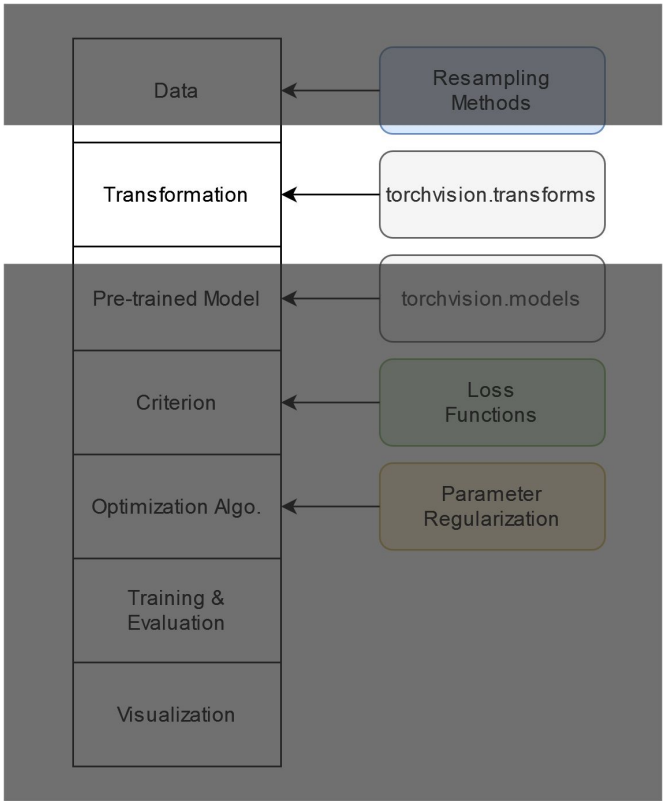
Data Transformations

```
from debiaser.general import transformer

transformer(image_size=224,
            normalize=(0.5,0.5),
            vflip=True,
            hflip=True,
            rotate=True,
            degrees=20,
            crop=False)
```

Before transformation

After transformation



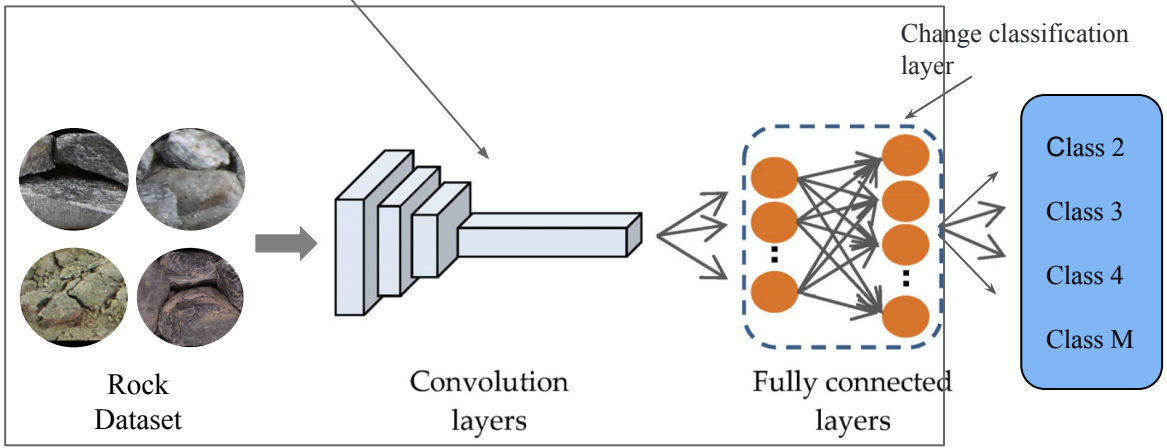


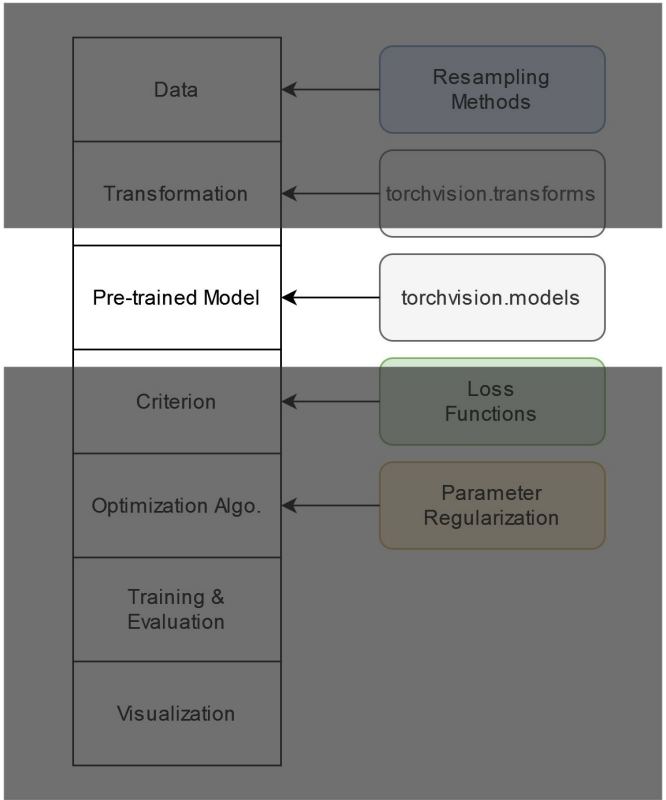
Pre-Trained Model

```
from debiaser.general import load_model

load_model(model_id='resnet18',
           n_class=4,
           pretrained=True,
           full=False)
```

ImageNet dataset

ResNet18
Mobilenet v2
EfficientNet B0





Loss Functions – Context

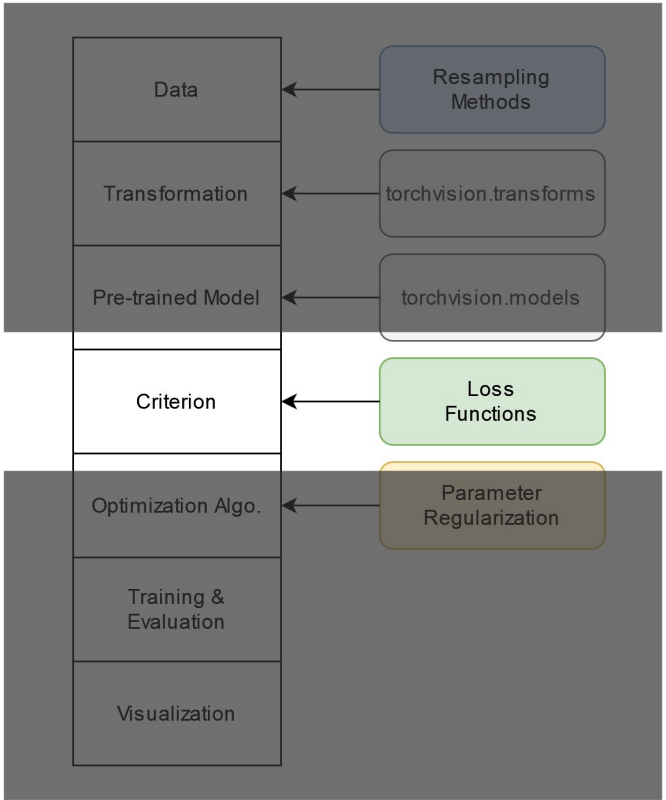
Problem:

- *A naive* model focuses on learning the characteristics of the majority class while, neglecting the examples from the minority → majority class contributes more to loss function

Solution:

- The key to addressing imbalanced datasets is:
  - Class-balanced loss via a weighting scheme (4 approaches)
  - make the loss function more sensitive to errors from the minority class (e.g. Mean False Error)

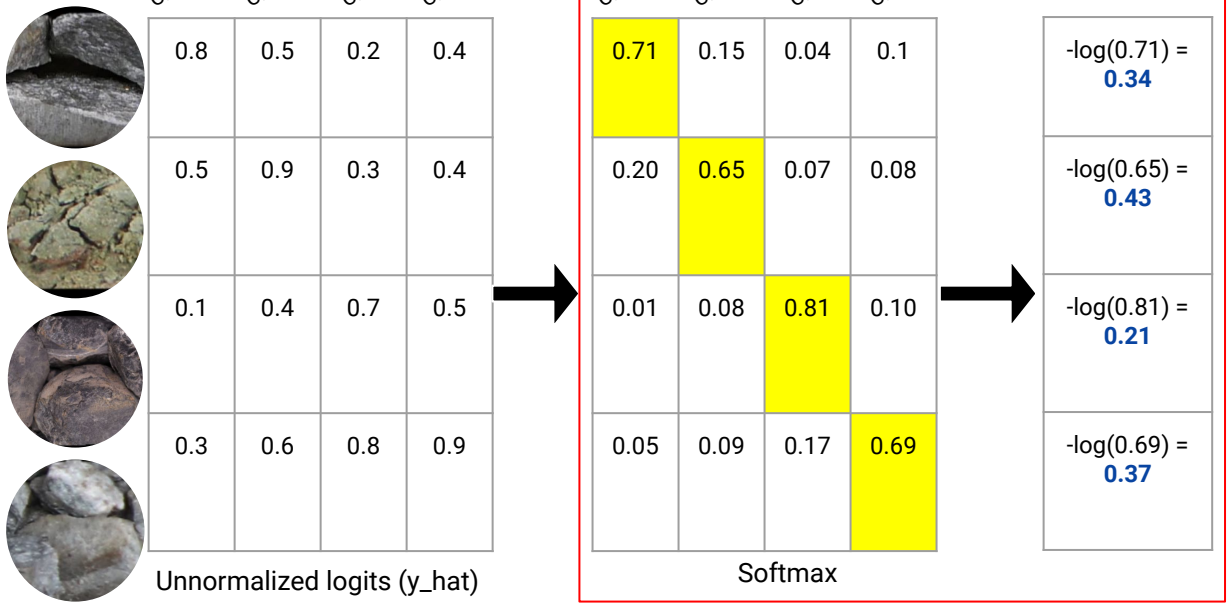
(ref)

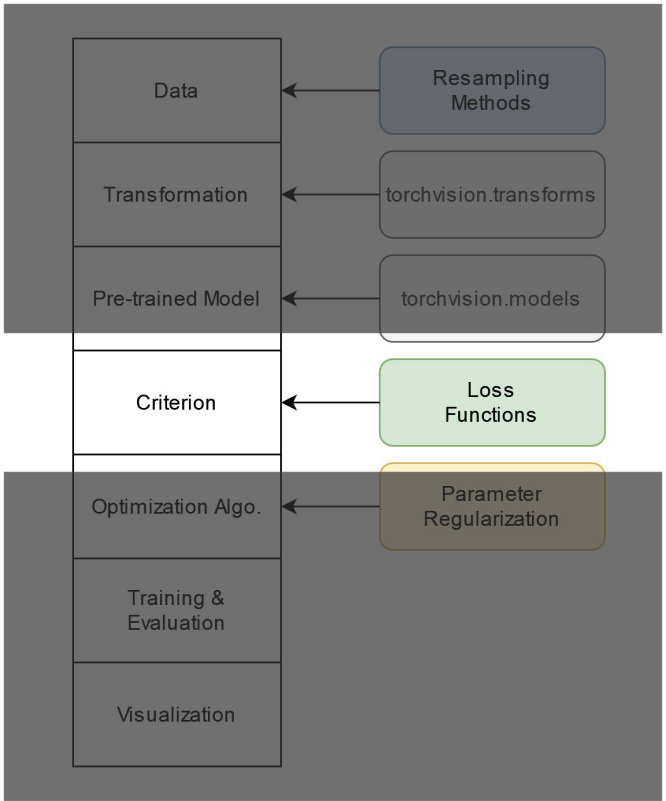


Loss Functions – Baseline Loss

```
from debiaser.general import load_loss

load_loss(loss_family='baseline',...)
```

= CrossEnropyLoss()





Loss Functions - Class-Balanced Loss

1) Pick a weighing scheme:

Inverse Number of Samples (INS)

$$w_{n,c} = \frac{1}{\text{Number of Samples in Class } c}$$

Inverse of Square Root of Number of Samples (ISNS)

$$w_{n,c} = \frac{1}{\sqrt[2]{\text{Number of Samples in Class } c}}$$

Inverse of Effective Number of Samples (ENS)

$$E_{n_c} = \frac{1-\beta^{n_c}}{1-\beta} \quad \beta \in [0, 1)$$

$$w_{n,c} = \frac{1}{E_{n_c}}$$

```
load_loss(loss_family='CB', loss_type=..., beta=0.999, gamma=2)
```

2) Use with the appropriate loss function:

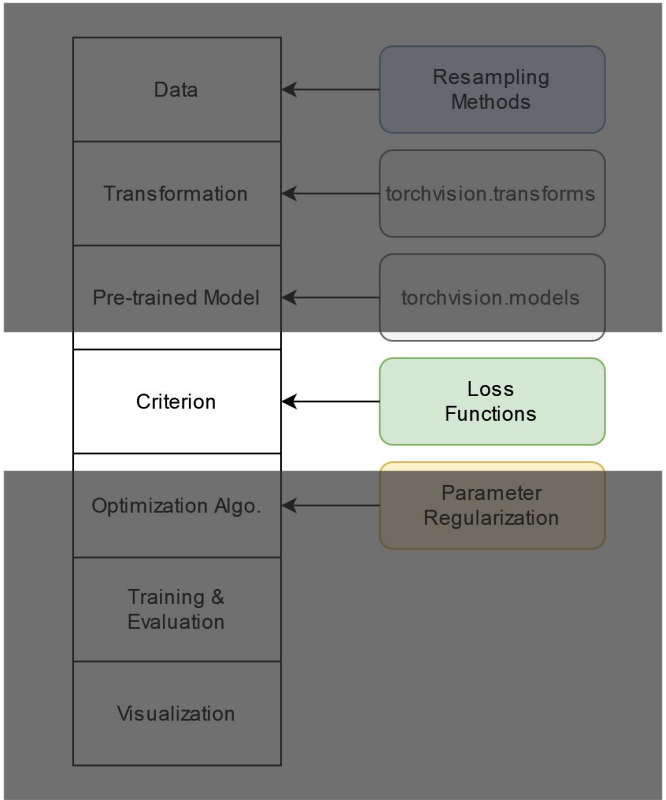
Weights * `softmax cross entropy`

Weights * `softmax binary cross entropy`

Weights * `sigmoid binary cross entropy`

Weights * `Focal Loss` = Weights* `modulator`*`sigmoid binary cross entropy`

(ref)



Loss Functions – Mean (Squared) False Error

```
load_loss(loss_family='MFE', loss_type='MSFE')
```

- Mean Squared Error (MSE):

$$l = \frac{1}{M}\Sigma_i \Sigma_n \frac{1}{2}(d_n^{(i)} - y_n^{(i)})^2$$

- Mean false error (MFE): A derivative of MSE

$$FPE = \frac{1}{N}\Sigma_{i=1}^{N} \Sigma_n \frac{1}{2}(d_n^{(i)} - y_n^{(i)})^2$$

$$FNE = \frac{1}{P}\Sigma_{i=1}^{P} \Sigma_n \frac{1}{2}(d_n^{(i)} - y_n^{(i)})^2$$

$$l' = FPE + FNE$$

- Mean squared false error (MSFE):

$$l'' = FPE^2 + FNE^2$$

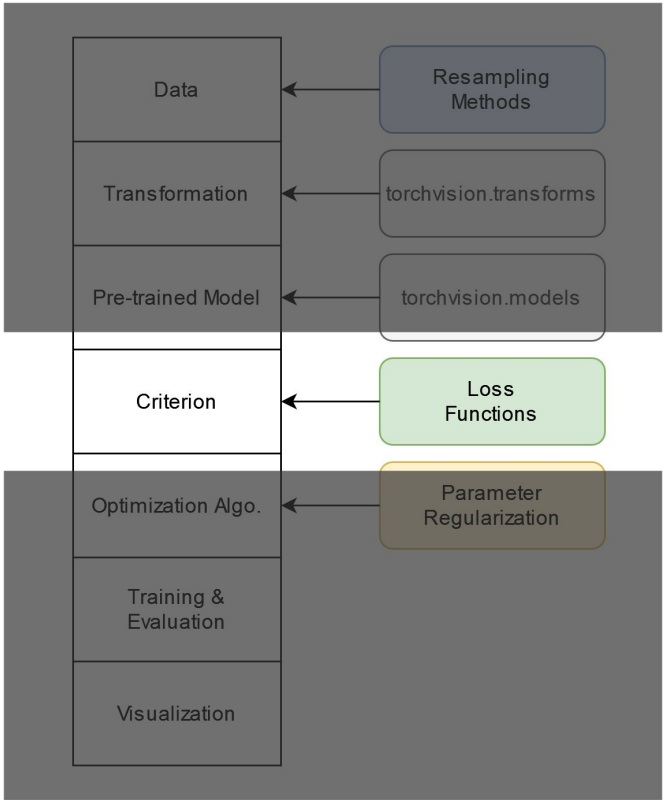
[(ref)]

N = sample size of negative class which is the majority class

P = sample size of positive class which is the minority class

d_n = the label vector

y_n = the prediction vector which is the output of sigmoid or softmax layer



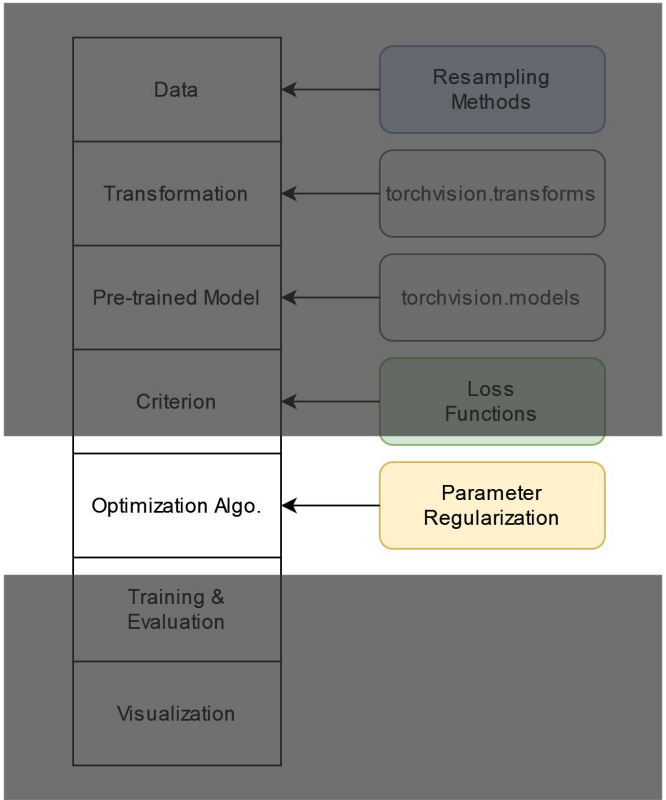
Parameter Regularization - Context

A naive model has large weights for common classes

- Solution: Try to balance the weights across classes for better predictions on the minority class!

```
from debiaser.general import load_optimizer


load_optimizer(dict_model,
               lr=1e-5,
               optimizer_name = 'Adam',
               strategy = None)
```

(ref)



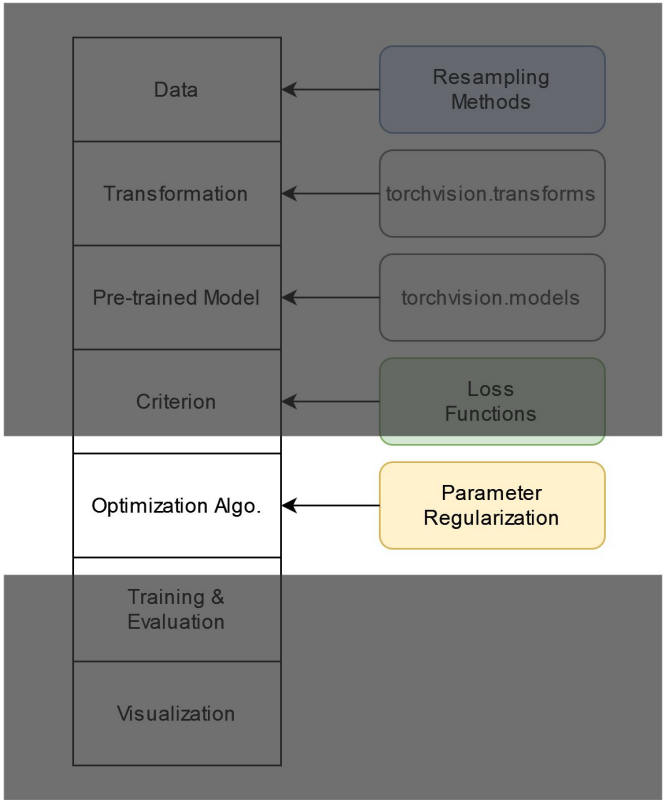
Parameter Regularization - Weight Decay

Weight decay penalizes larger weights more severely than smaller weights

- More balanced weight growth
- Decreases model complexity
- Improves generalization

```
load_optimizer(dict_model,
               lr=1e-5,
               optimizer_name='Adam',
               strategy='wei_dec',
               weight_decay=0.005)
```

(ref)

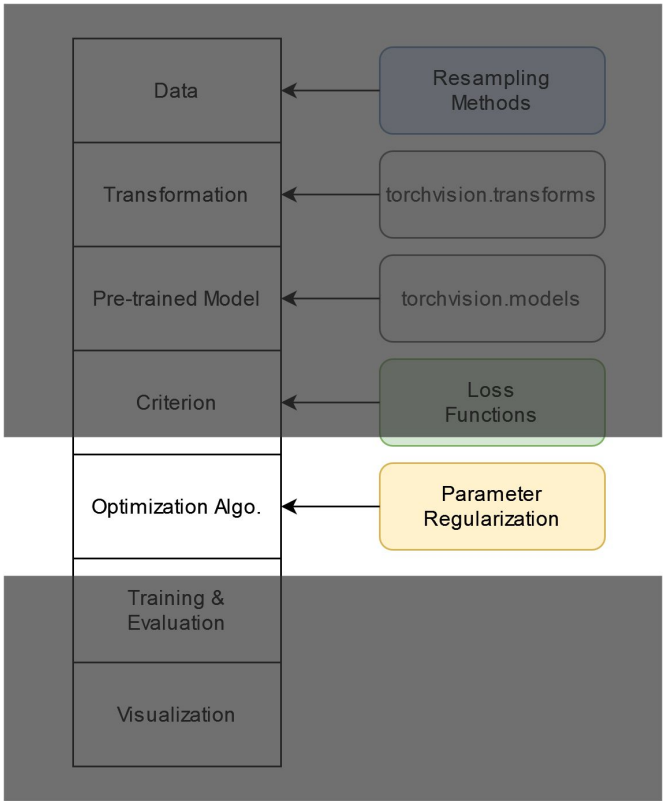


Parameter Regularization - L2-Norm

L2- normalization places a strict restriction on the weights.

- Improves minority class performance
- Sacrifices overall model performance

```
load_optimizer(dict_model,
               lr=1e-5,
               optimizer_name='Adam',
               strategy='l2_norm')
```



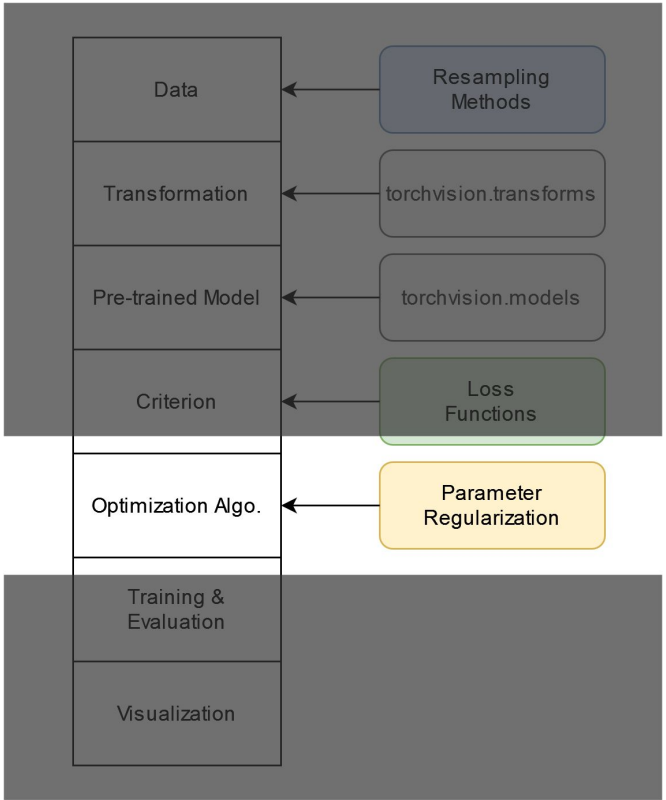
(ref)

Parameter Regularization - MaxNorm

MaxNorm encourages smaller weight growth and places a limit on weight norms (δ)

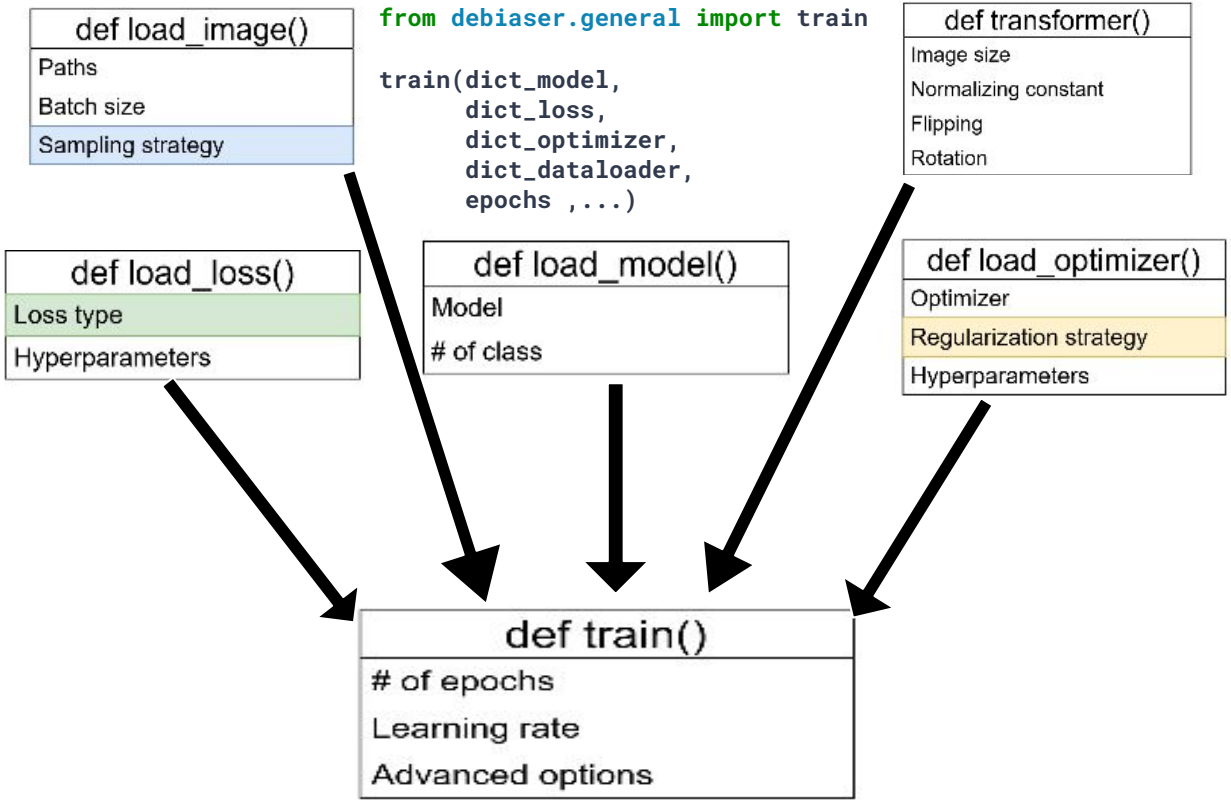
- Weights are stable even at high learning rates because they are always bounded.

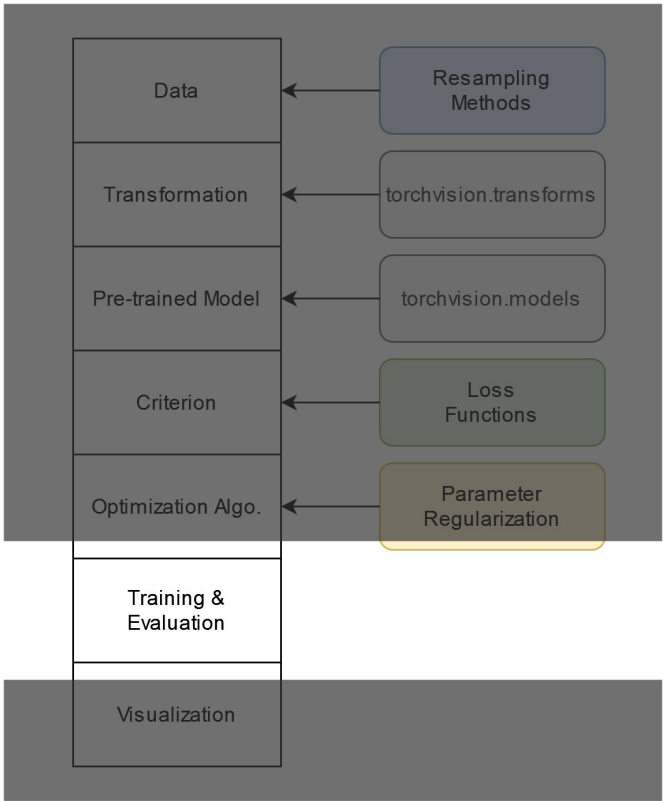
```
load_optimizer(dict_model,
               lr=1e-5,
               optimizer_name='Adam',
               strategy='maxnorm',
               thresh=0.8)
```



(ref)

Training


def load_image()
Paths
Batch size
Sampling strategy
from debiaser.general import train
train(dict_model,
dict_loss,
dict_optimizer,
dict_dataloader,
epochs ,...)
def transformer()
Image size
Normalizing constant
Flipping
Rotation
def load_loss()
Loss type
Hyperparameters
def load_model()
Model
# of class
def load_optimizer()
Optimizer
Regularization strategy
Hyperparameters
def train()
# of epochs
Learning rate
Advanced options


Data
Resampling Methods
Transformation
torchvision.transforms
Pre-trained Model
torchvision.models
Criterion
Loss Functions
Optimization Algo.
Parameter Regularization
Training & Evaluation
Visualization

Introduction & Objectives

Data Science Techniques

Data Product

Results

Conclusions and Recommendations

User Persona

- Highly technical
- Familiar with Python and programming

- **Generalized!**
- **Modify easily!**
- **JSON formats!**

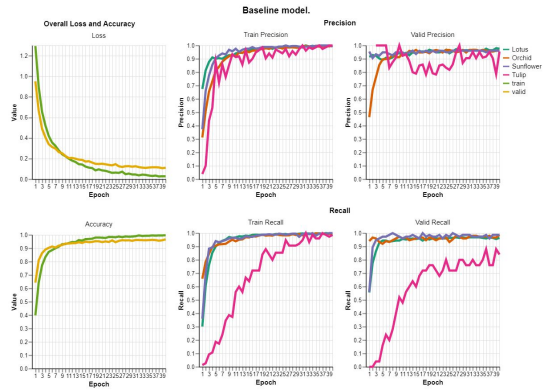
Basic functions

load_image transformer load_model load_loss load_optimizer train

```
{
    'batch_size': 32,
    'model': 'resnet18',
    'loss_type': 'focal',
    'beta':0.9999,
    ...
}
```

Pipeline

.json



Json File Schema

- Json
  - **note**
  - **date**
  - results
  - y
  - var_dict

```
{
    ‘note’: ‘baseline-resnet’,
    ‘date’: ‘20230609-122303,
    ...
}
```

Json File Schema

- Json
  - note
  - date
  - **results**
    - Training and validation set
    - Loss and accuracy
    - Precision and recall for each class
  - y
  - var_dict

```
{
    'results': {
        'train_loss': [...],
        'train_accuracy': [...],
        ...
    },
    ...
}
```

Json File Schema

- Json
  - note
  - date
  - results
  - **y**
    - All target labels & predicted labels
    - Both training and validation set
    - All epochs
  - var_dict

```
{
    'y': {
        'train': {
            'true': {
                0: [1, 0, 1, ...],
                1: [0, 0, 0, ...],
                ...
            },
            'pred' {...},
        },
        'valid':{...}
}
```

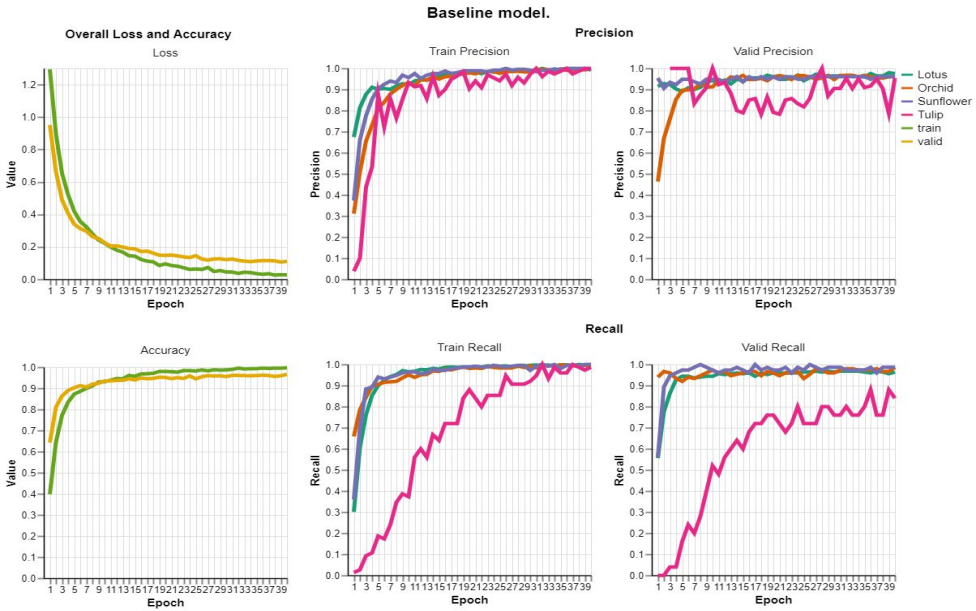
Json File Schema

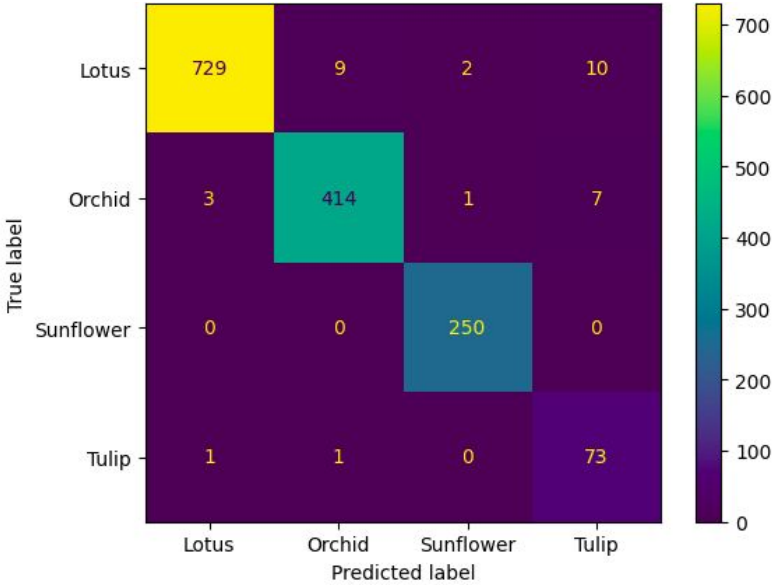
- Json
  - note
  - date
  - results
  - y
  - **var_dict**
    - All parameters used
    - Identifiers for trials
    - Useful for comparing trials

```
{
    ‘var_dict’: {
        ‘batch_size’: 32,
        ‘sampling_strategy’: None,
        ‘model_name’: ‘resnet18’,
        ‘loss_family’: ‘CB’,
        ‘loss_type’: ‘focal’,
        ‘beta’: 0.999,
        ‘gamma’: 0.5,
        ‘optimizer’: ‘Adam’,
        ‘learning_rate’: 1e-5,
        ‘epochs’: 50
}
```

Visualizing One JSON file

.json → visualization.py





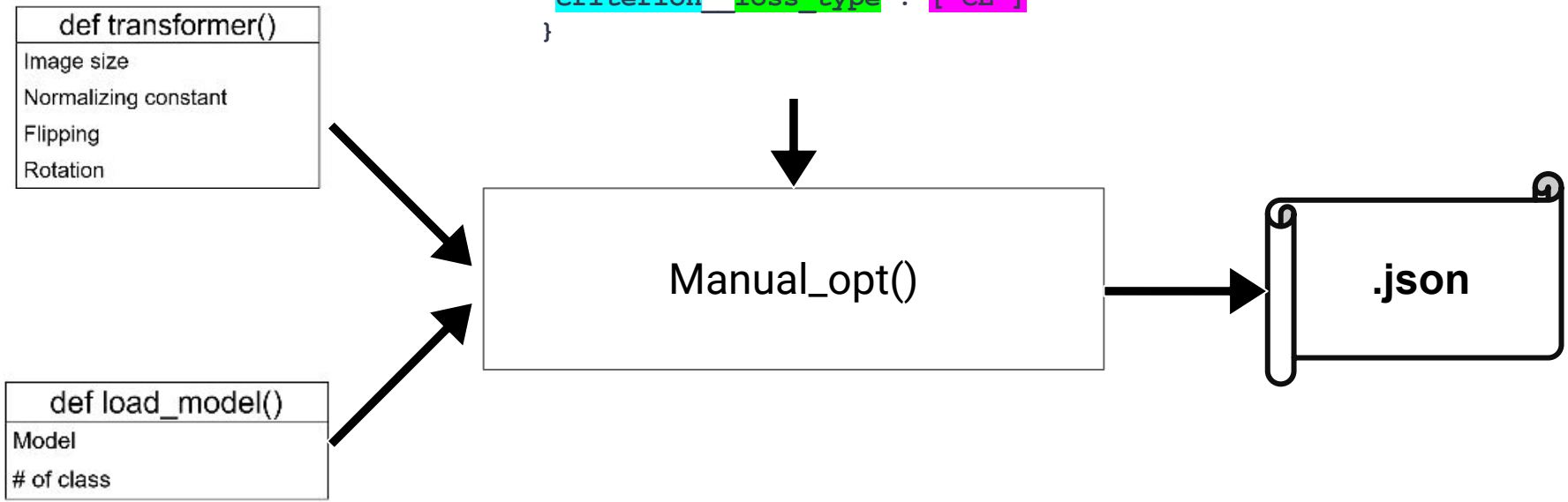
Hyperparameter Optimization

GridSearch with K-fold cross validation for each combination of parameters

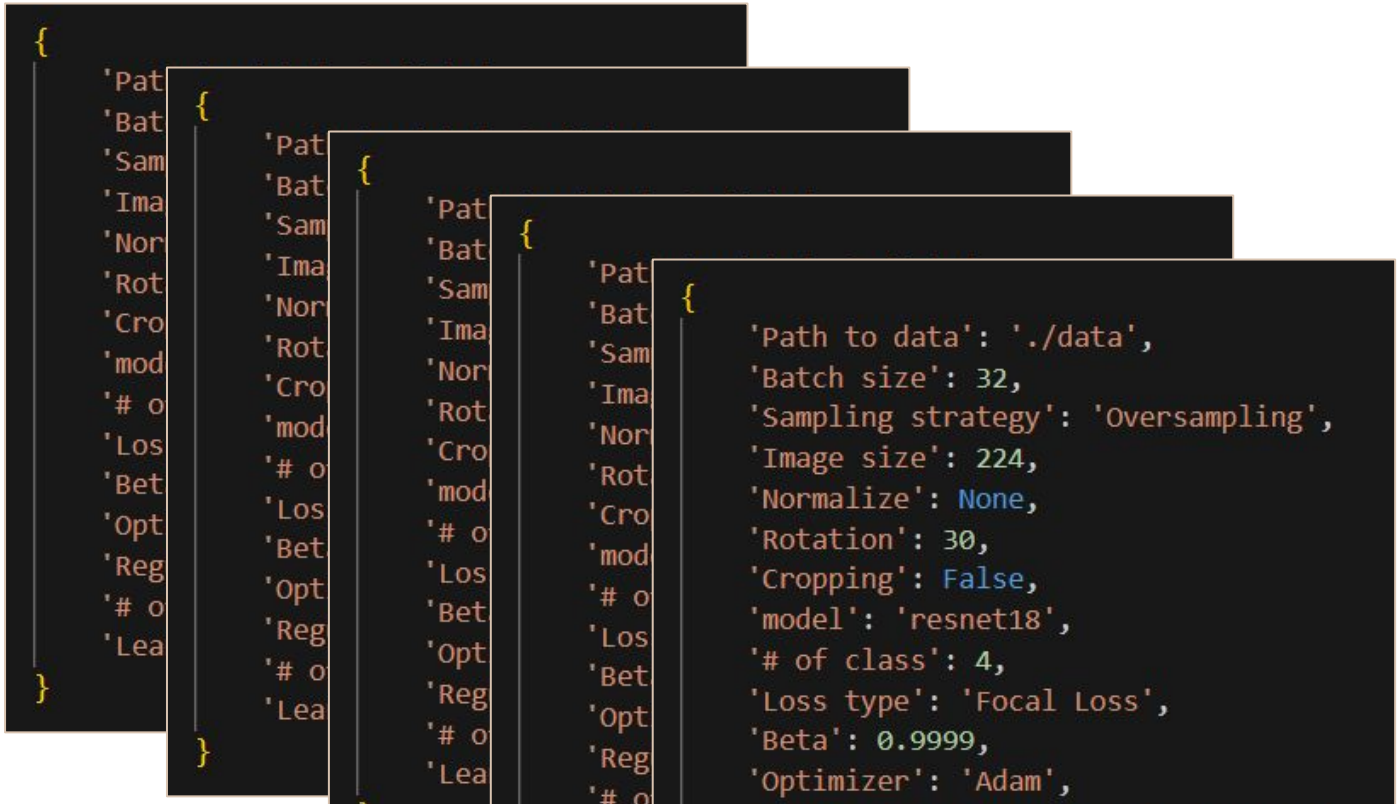
Optimizer	Loss Function	Train
Learning rate Optimizer Regularization strategy and associated parameters	Loss family Loss type Associated parameters	Number of epochs Learning rate scheduler Gradient clipping value Patience tolerance

Hyperparameter Optimization

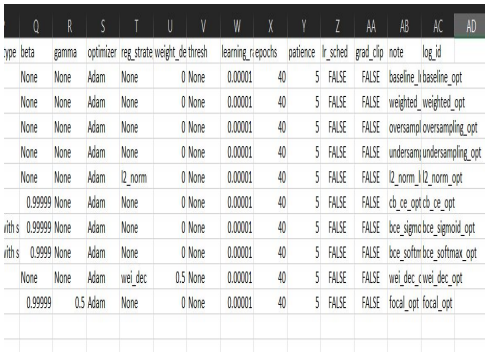
```
params = {
        'criterion__loss_family':["CB"],
        'criterion__beta': [0.5, 0.9],
        'criterion__loss_type': ["CE"]
        }
```



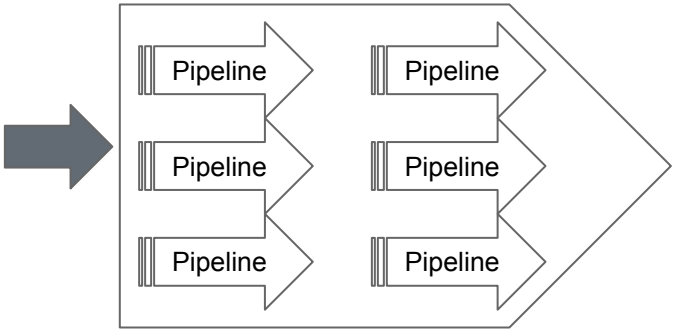
Automation and Reproducibility

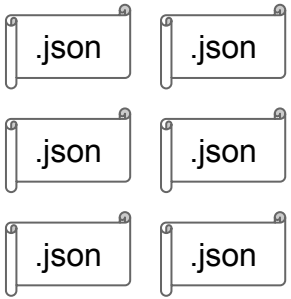


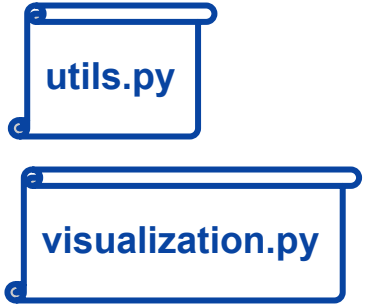
Automation and Reproducibility



Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
ype beta	gamma	optimizer	reg_strate	weight_de	thresh	learning_r	epochs	patience	lr_sched	grad_clip	note	log_id	
None	None	Adam	None	0	None	0.00001	40	5	FALSE	FALSE	baseline_l	baseline_opt	
None	None	Adam	None	0	None	0.00001	40	5	FALSE	FALSE	weighted_	weighted_opt	
None	None	Adam	None	0	None	0.00001	40	5	FALSE	FALSE	oversampl	oversampling_opt	
None	None	Adam	None	0	None	0.00001	40	5	FALSE	FALSE	undersamp	undersampling_opt	
None	None	Adam	l2_norm	0	None	0.00001	40	5	FALSE	FALSE	l2_norm_l	l2_norm_opt	
0.99999	None	Adam	None	0	None	0.00001	40	5	FALSE	FALSE	cb_ce_opt	cb_ce_opt	
ith s 0.99999	None	Adam	None	0	None	0.00001	40	5	FALSE	FALSE	bce_sigmc	bce_sigmoid_opt	
ith s 0.9999	None	Adam	None	0	None	0.00001	40	5	FALSE	FALSE	bce_softm	bce_softmax_opt	
None	None	Adam	wei_dec	0.5	None	0.00001	40	5	FALSE	FALSE	wei_dec_c	wei_dec_opt	
0.99999	0.5	Adam	None	0	None	0.00001	40	5	FALSE	FALSE	focal_opt	focal_opt	







Data product: Strengths & Weaknesses

- Python package
- High flexibility
- Easy to make improvements
- Generalized
- Automated

- Not friendly to non-technical users
- Not covering all methods
- Implemented methods can only be used within the designed system

Introduction & Objectives

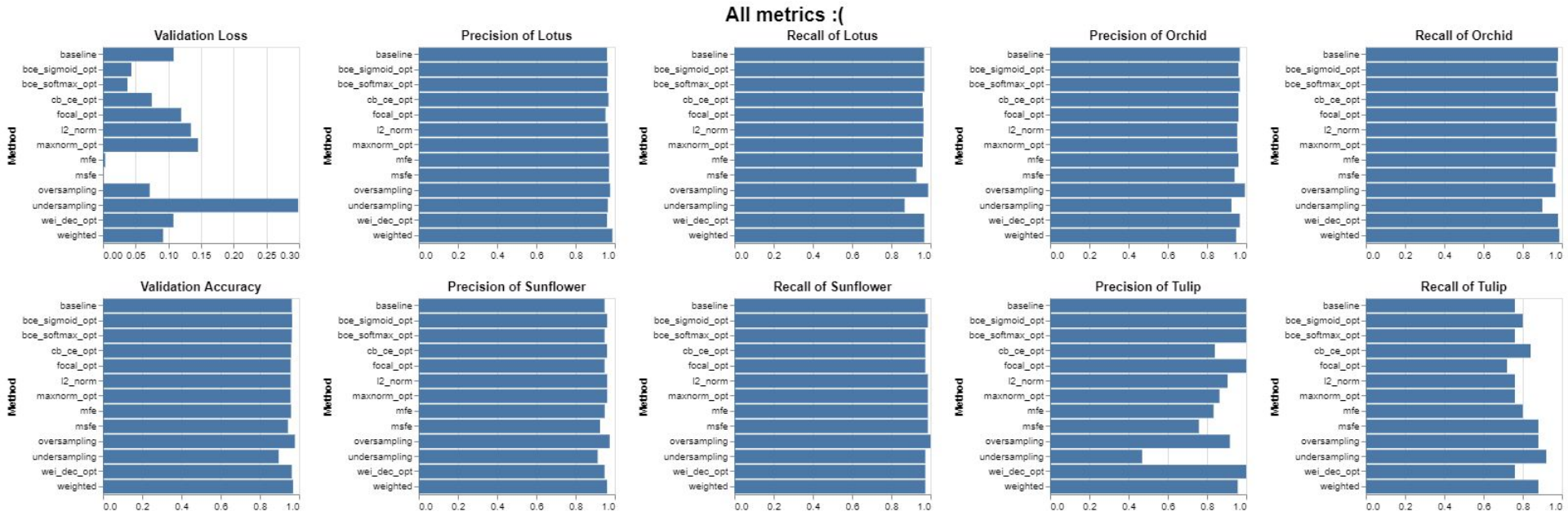
Data Science Techniques

Data Product

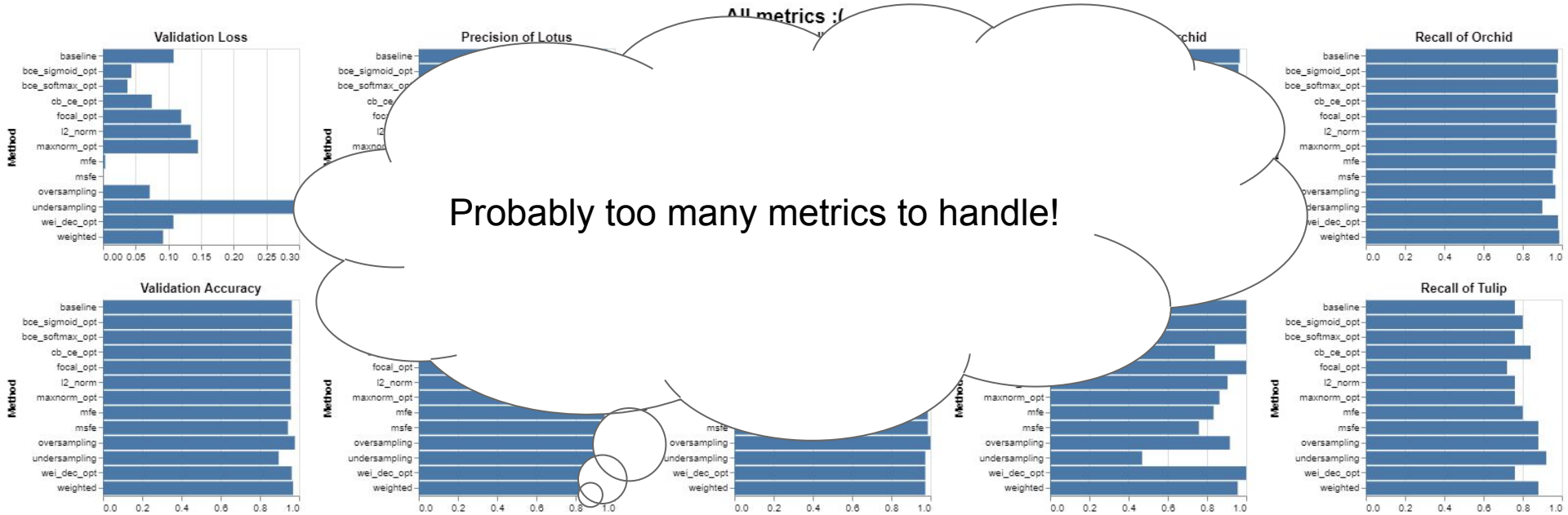
Results

Conclusions and Recommendations

Decision-making on “Best”



Decision-making on “Best”



Traditional Weighted Metrics

Precision	*n*
0.7	30
0.8	20
0.3	10

$$\text{macro average} = \frac{0.7 + 0.8 + 0.3}{3} = 0.6$$

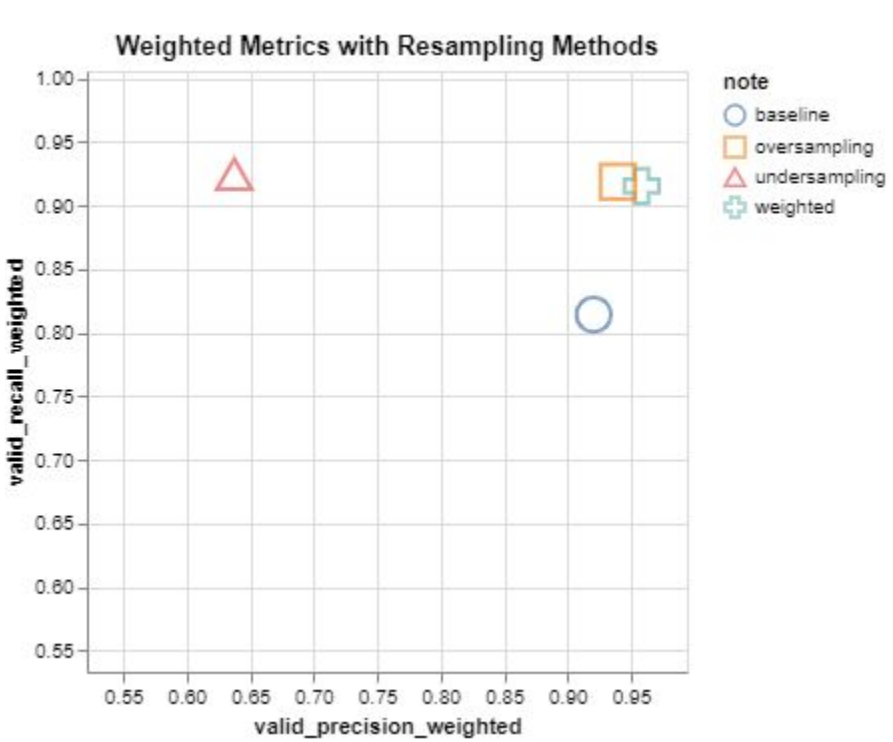
$$\text{weighted average} = \frac{0.7 \times 30 + 0.8 \times 20 + 0.3 \times 10}{30 + 20 + 10} = 0.6667$$

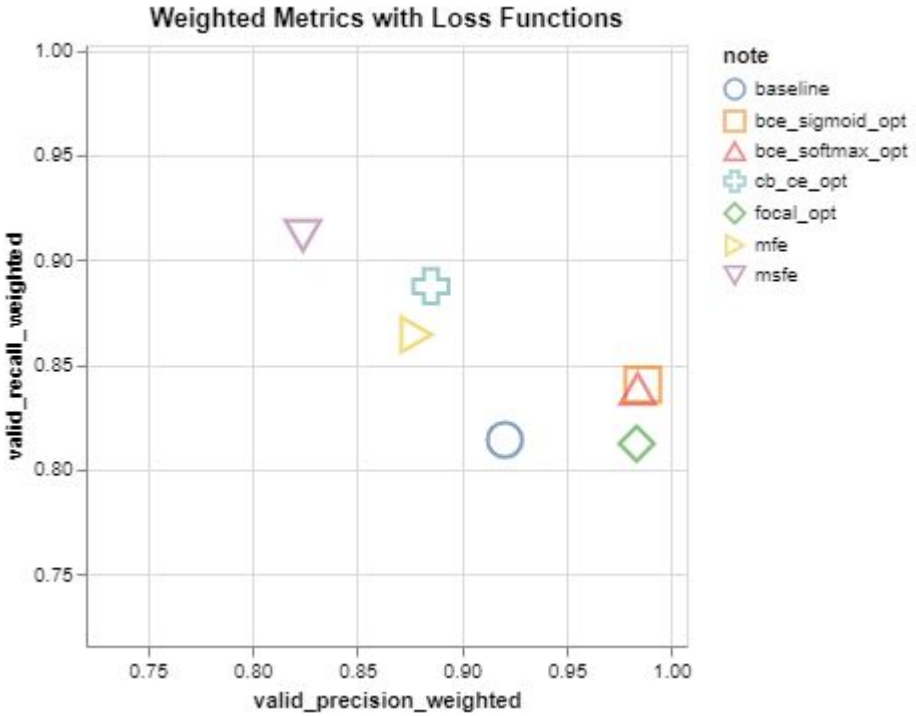
Weighted Metrics

Precision	*n*	1/*n*
0.7	30	1/30
0.8	20	1/20
0.3	10	1/10

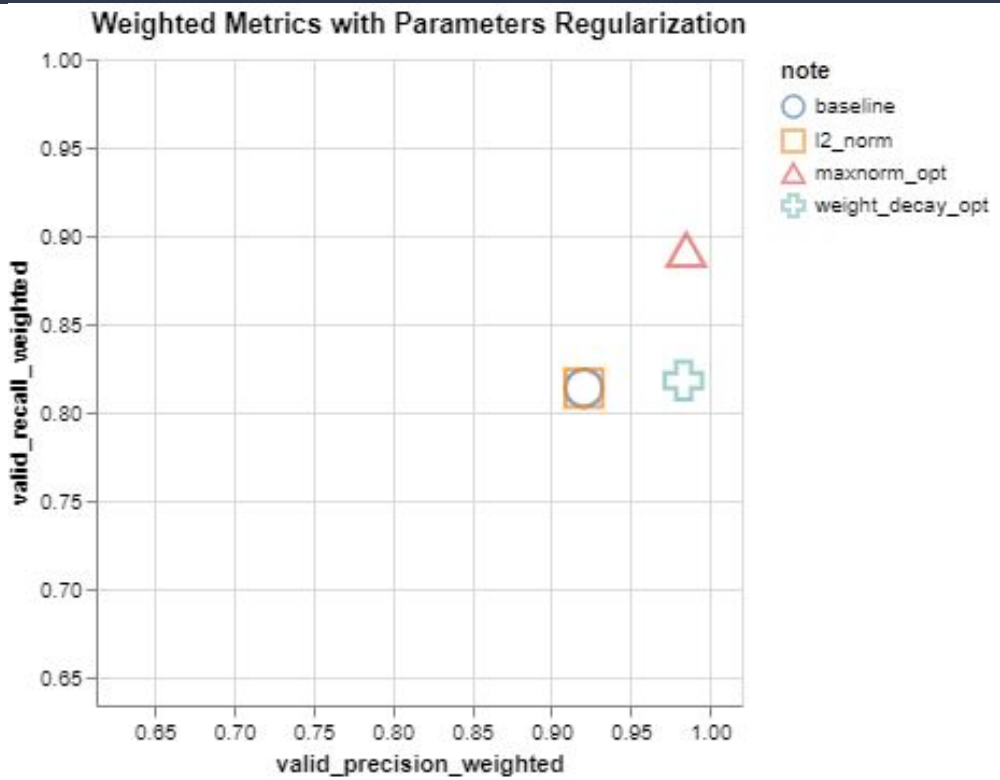
$$\text{new weighted average} = \frac{0.7 \times 1/30 + 0.8 \times 1/20 + 0.3 \times 1/10}{1/30 + 1/20 + 1/10} = 0.4$$

Results

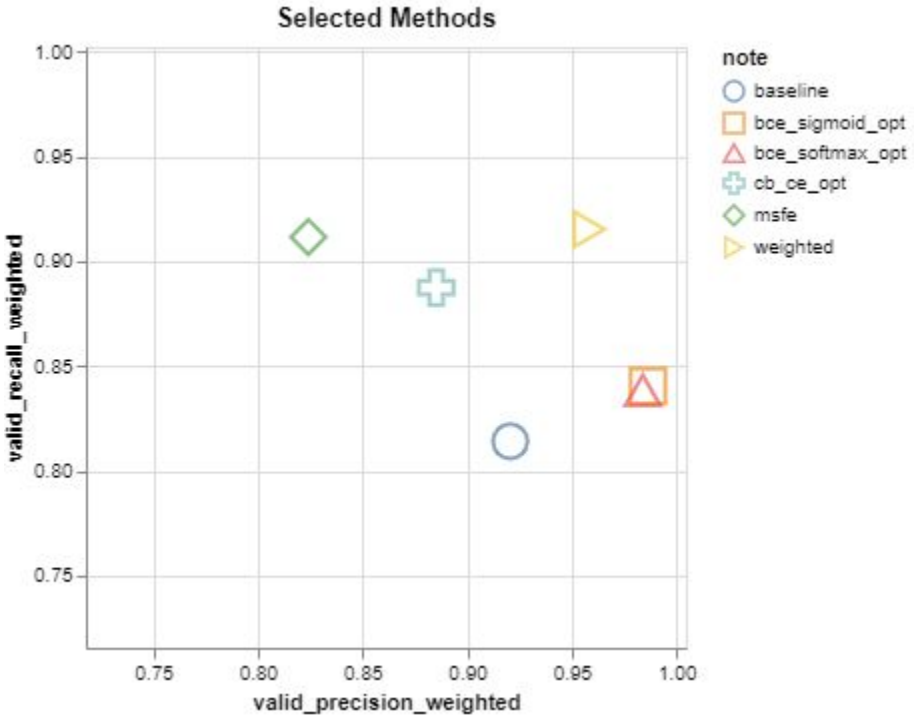

Weighted Metrics with Resampling Methods
valid_recall_weighted
valid_precision_weighted
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95
note
baseline
oversampling
undersampling
weighted


Weighted Metrics with Loss Functions
valid_recall_weighted
valid_precision_weighted
1.00
0.95
0.90
0.85
0.80
0.75
0.75 0.80 0.85 0.90 0.95 1.00
note
baseline
bce_sigmoid_opt
bce_softmax_opt
cb_ce_opt
focal_opt
mfe
msfe

Results (cont.)



Results: Best Methods



Introduction & Objectives

Data Science Techniques

Data Product

Results

Conclusions and Recommendations

Conclusions and Recommendations

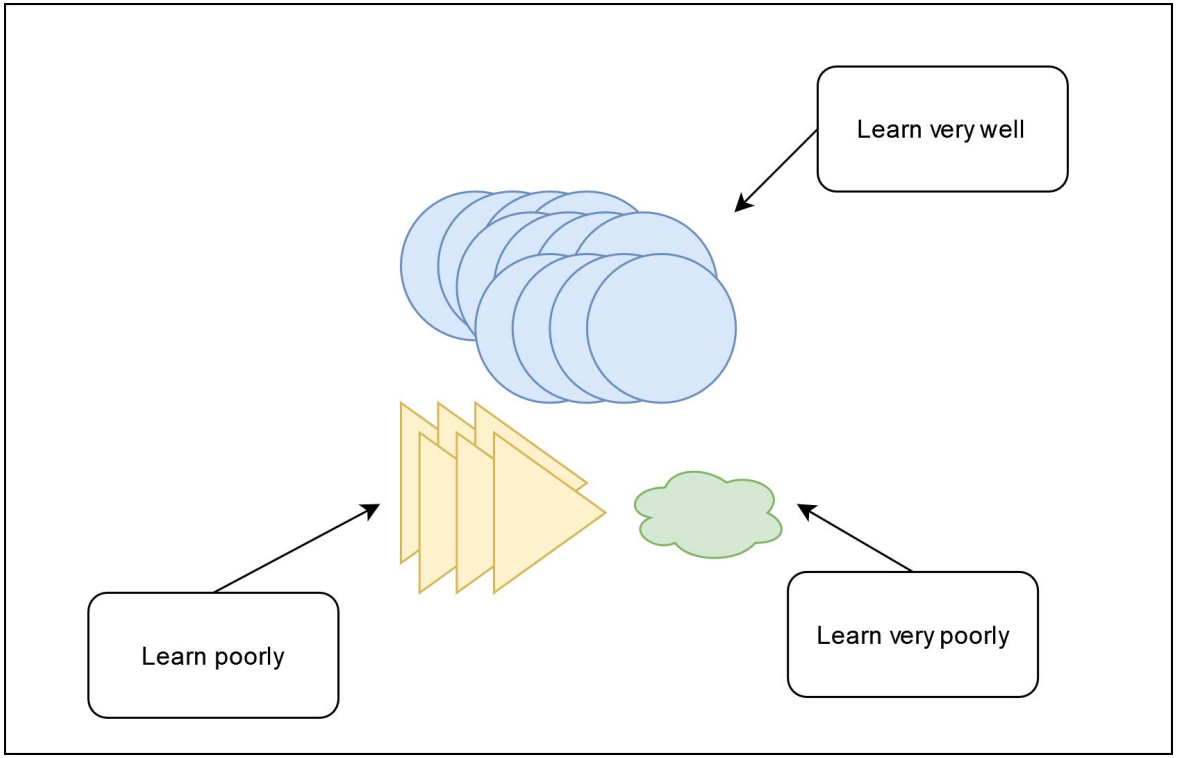
Created a package that is able to train and compare CNN models that can employ various techniques to tackle data imbalance problems

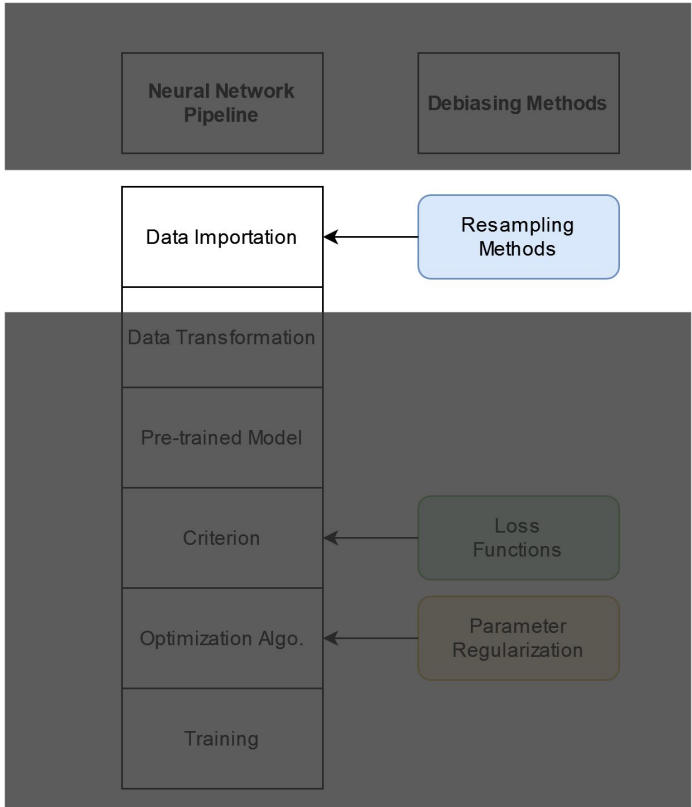
- ★ Successfully implemented various methods involving resampling techniques, parameter regularization, and custom loss functions to improve precision/recall and accuracy in the minority class
- ★ May be limited by data quality / data preprocessing steps
- ★ Add more features:
  - ○ Adapting to more pytorch pretrained models
  - ○ Exploring different weighing schemes for class-balanced loss
  - ○ ROC curve analysis

Thank you!

Any questions?

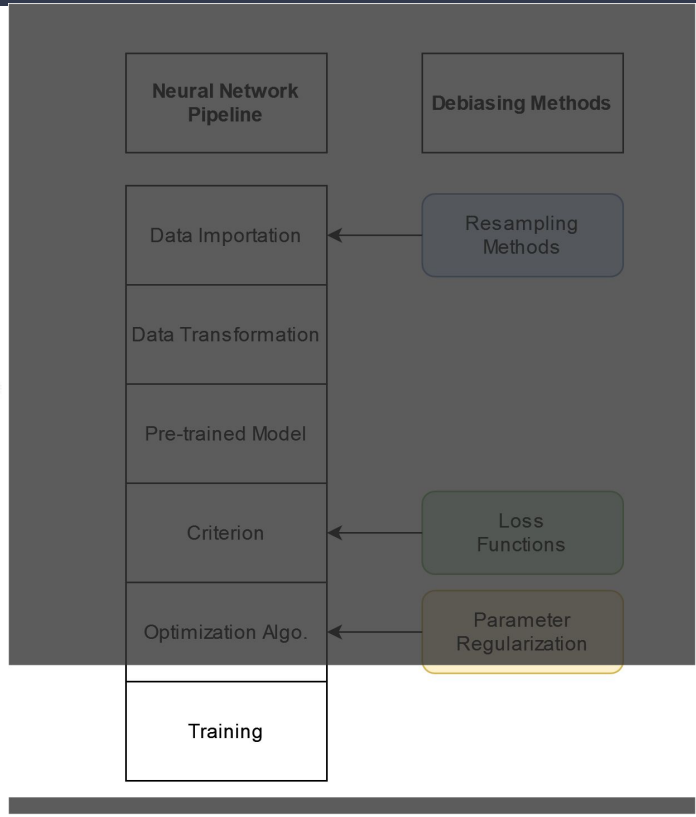
Resampling


Learn very well
Learn poorly
Learn very poorly

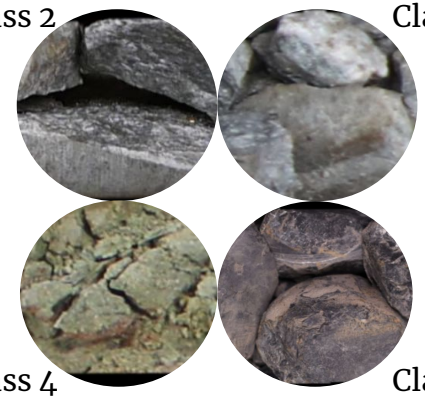

Neural Network Pipeline
Debiasing Methods
Data Importation
Resampling Methods
Data Transformation
Pre-trained Model
Criterion
Loss Functions
Optimization Algo.
Parameter Regularization
Training

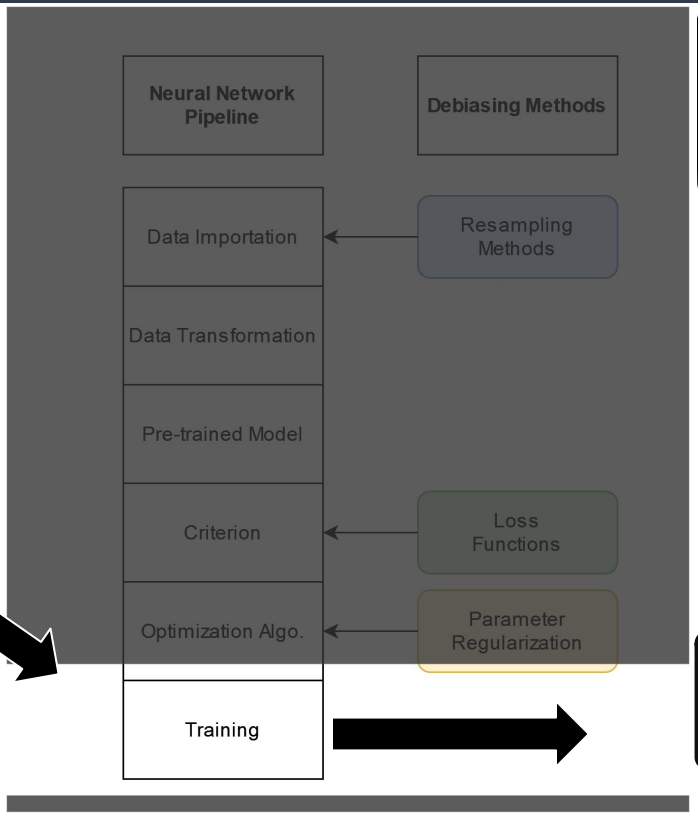
Training

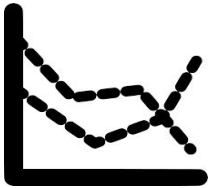
```
dict_transform = transformer(image_size=224,
                             normalize=(0.5,0.5),
                             vflip=True,
                             hflip=True,
                             rotate=True,
                             degrees=20,
                             crop=True)

dict_model = load_model(model_id='resnet18',
                        n_class=4,
                        pretrained=True,
                        full=False)

dict_dataloader = load_image(train_path='/data/train/',
                             valid_path='/data/validation/',
                             batch_size=64,
                             dict_transform=dict_transform,
                             sampling_strategy=None)

dict_optimizer = load_optimizer(dict_model=dict_model,
                                lr=1e-6,
                                optimizer_name='SGD',
                                strategy=None,
                                weight_decay=0.001,
                                thresh=1.0)

dict_loss = load_loss(dict_dataloader=dict_dataloader,
                      loss_family='CB',
                      loss_type='CE',
                      beta=0.999,
                      gamma=None)

results = train(dict_model=dict_model,
                dict_loss=dict_loss,
                dict_optimizer=dict_optimizer,
                dict_dataloader=dict_dataloader,
                epochs=20,
                verbose=False,
                note='Testing - ResNet',
                log_id='Resnet-Rocks-lr=1e-6')
```

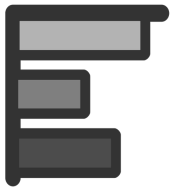


Training





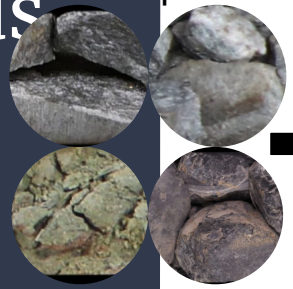




Actual Class \ Predicted Class	Yes	No
Yes	TP	FN
No	FP	TN

Debiasing Methods Overview

Input



Output

Pipeline

Neural Network Pipeline	Debiasing Methods
Data Importation	Resampling Methods
Data Transformation	
Pre-trained Model	PyTorch Image
Criterion	Loss Functions
Optimization Algo.	Parameter Regularization
Training	

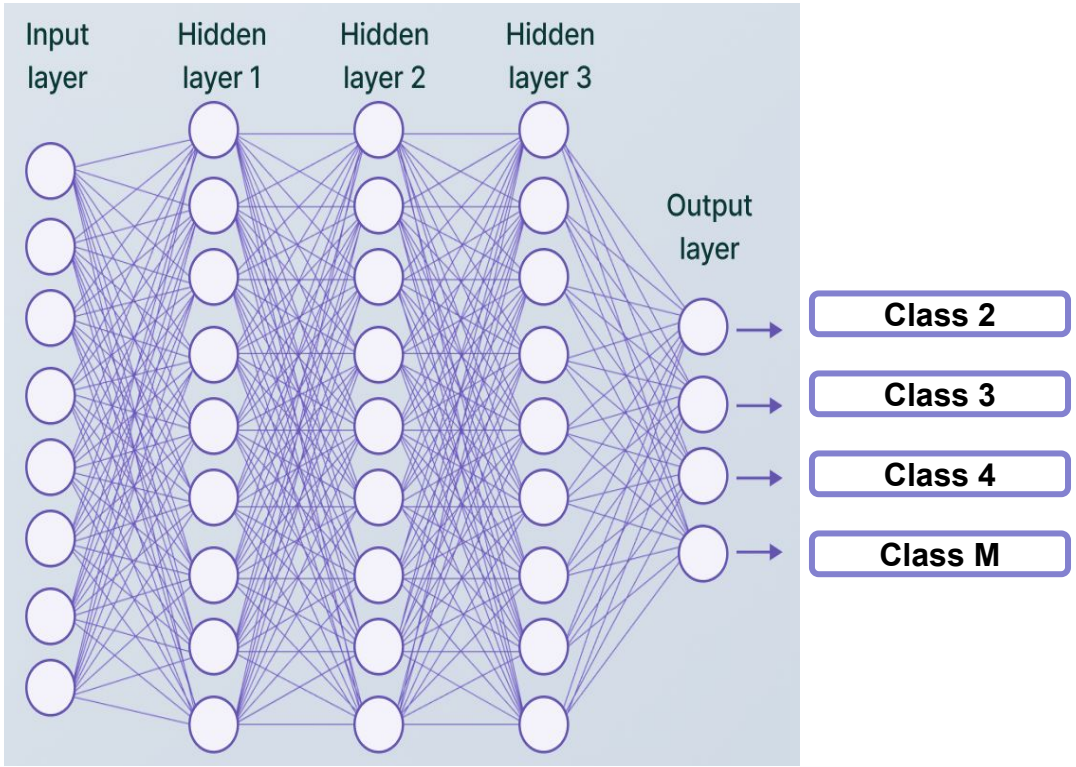
General CNN Structure

Class 2

Class 3

Class 4

Class M



ResNet18

layer name	output size	18-layer
conv1	112×112	
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2
	1×1	
FLOPs		1.8×10^9

Basic Block

Number of times to repeat the Basic Block

Changed FC Layer from:

```
(fc): Linear(in_features=512, out_features=1000, bias=True)
```

to:

```
new_layers = nn.Sequential(
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(512, n_class)
)
```

MobileNet v2

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Changed classifier from:

```
(classifier): Sequential(
  (0): Dropout(p=0.2, inplace=False)
  (1): Linear(in_features=1280, out_features=1000, bias=True)
)
```

to:

```
new_layers = nn.Sequential(
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(1280, n_class)
)
```

EfficientNet B0

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

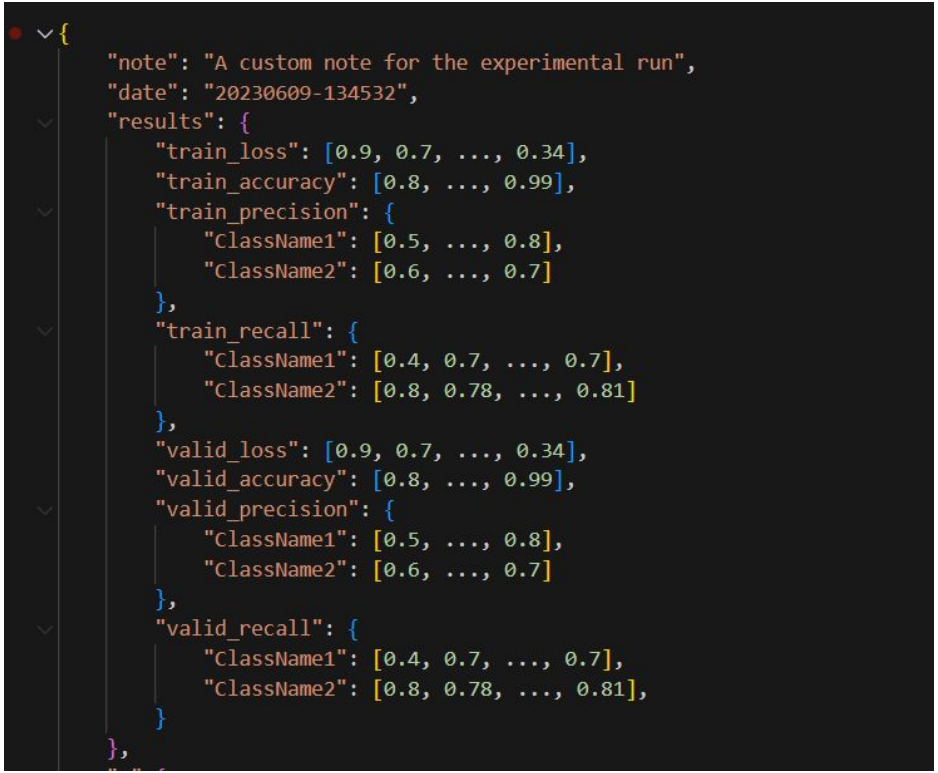
Changed classifier from:

```
(classifier): Sequential(
  (0): Dropout(p=0.2, inplace=True)
  (1): Linear(in_features=1280, out_features=1000, bias=True)
)
)
```

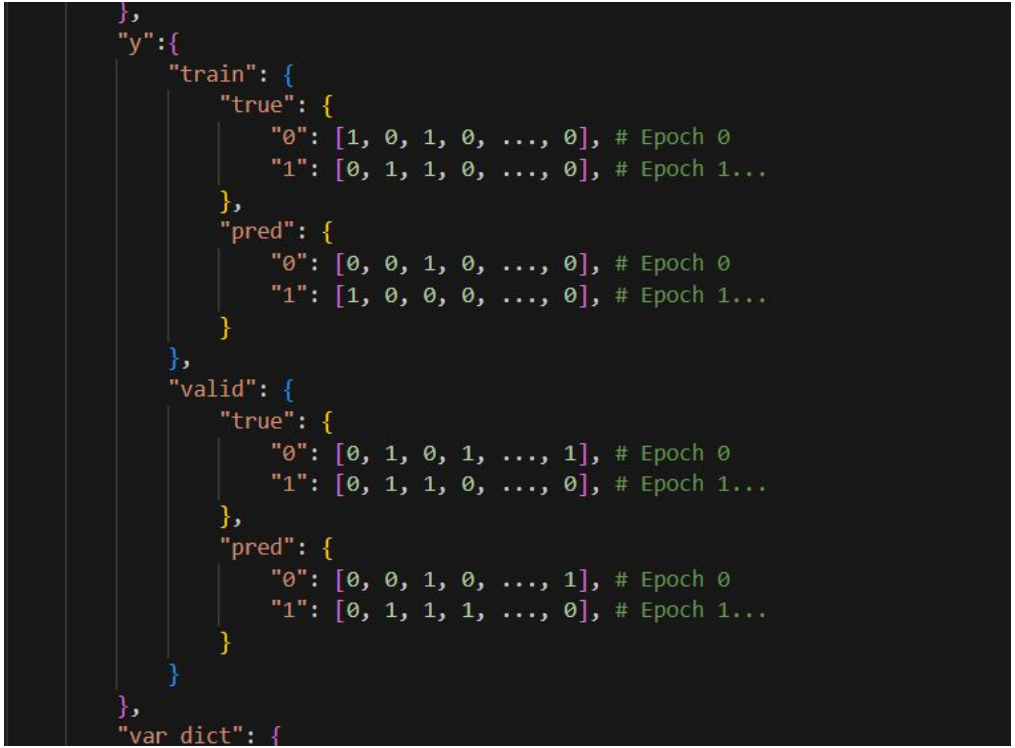
to:

```
new_layers = nn.Sequential(
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(1280, n_class)
)
```

JSON File Format



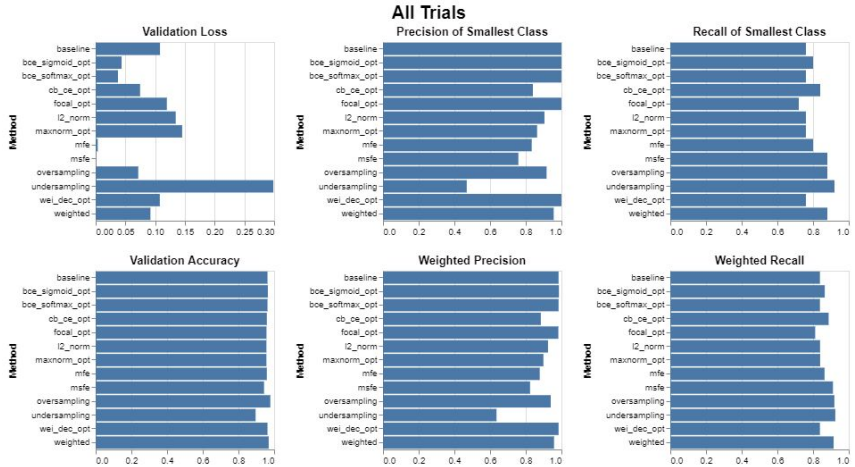
JSON File Format

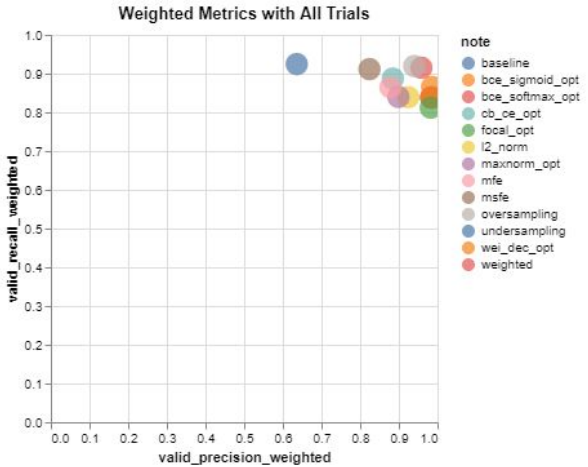


JSON File Format

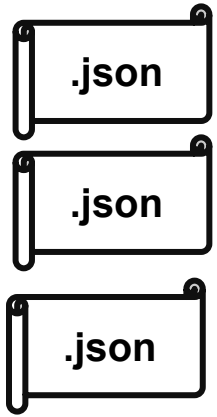
Statistics and Visualization

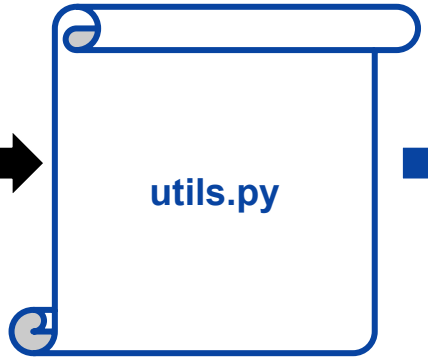
	valid_loss	valid_accuracy	sampling_strategy	loss_family	loss_type	reg_strategy	learning_rate	note
2	0.071542	0.978516	oversampling	baseline	None	None	1.00e-05	oversampling_opt
1	0.091995	0.969531	weighted	baseline	None	None	1.00e-05	weighted_opt
6	0.043558	0.964844	None	CB	BCE with sigmoid	None	1.00e-05	bce_sigmoid_opt
0	0.108101	0.962891	None	baseline	None	None	1.00e-05	baseline_opt





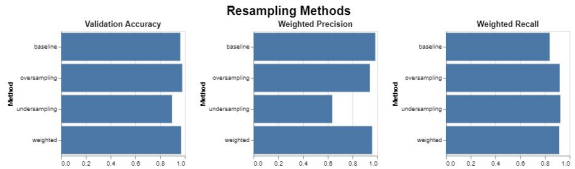
Graphing Multiple JSON files

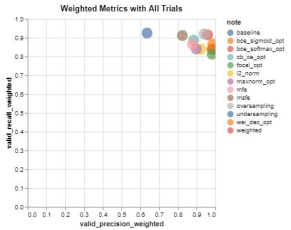


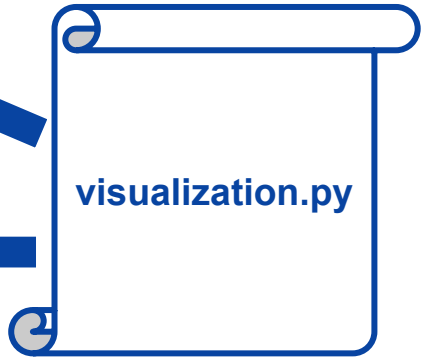


dataframe

	valid_loss	valid_accuracy	sampling_strategy	loss_family	loss_type	reg_strategy	learning_rate	note
2	0.071542	0.978516	oversampling	baseline	None	None	1.00e-05	oversampling_opt
1	0.091995	0.969531	weighted	baseline	None	None	1.00e-05	weighted_opt
6	0.043558	0.964844	None	CB	BCE with sigmoid	None	1.00e-05	bce_sigmoid_opt
0	0.108101	0.962891	None	baseline	None	None	1.00e-05	baseline_opt







Loss Functions – Effective Number

$E_n = (1 - \beta^n)/(1 - \beta)$, where $\beta \varepsilon$ [0, 1):

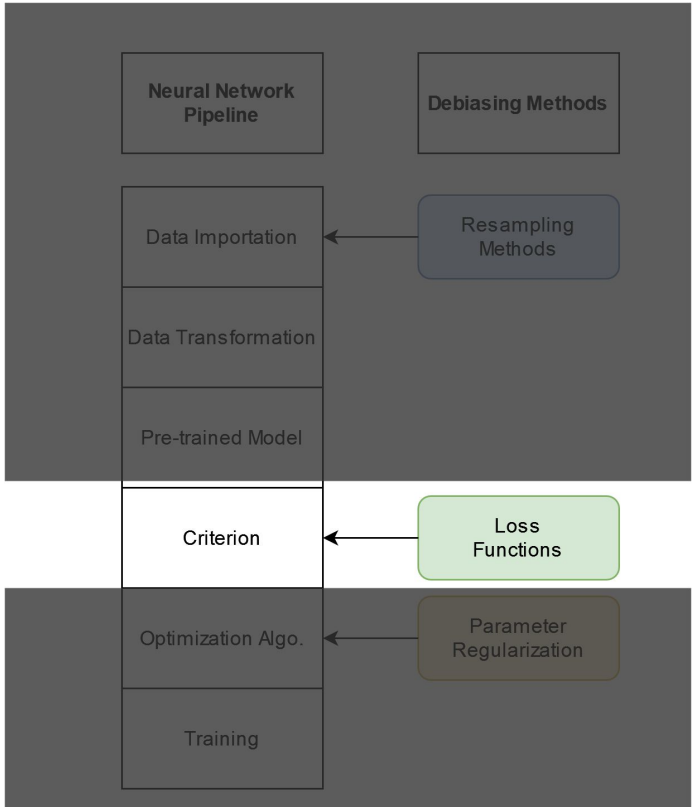
- β = 0 : no re-weighing
- β = 1 : re-weighing by inverse class frequencies

So, class-balanced (CB) loss will be defined as:

$$\mathrm{CB}(\mathbf{p}, y) = \frac{1}{E_{n_y}} \mathcal{L}(\mathbf{p}, y) = \frac{1-\beta}{1-\beta^{n_y}} \mathcal{L}(\mathbf{p}, y)$$

Where **p** is class probabilities, *y* is ground-truth labels, n_y is the number of samples in the ground-truth class and $\mathcal{L}$ (**p**,y) is any loss function.

(ref)



Loss Functions - Class-Balanced Loss

$$\mathrm{CB}_{\mathrm{softmax}}(\mathbf{z}, y) = -\frac{1-\beta}{1-\beta^{n_y}} \log\left(\frac{\exp(z_y)}{\sum_{j=1}^{C}\exp(z_j)}\right)$$

cross_entropy(raw_logits, labels)

binary_cross_entropy(softmax_output, labels)

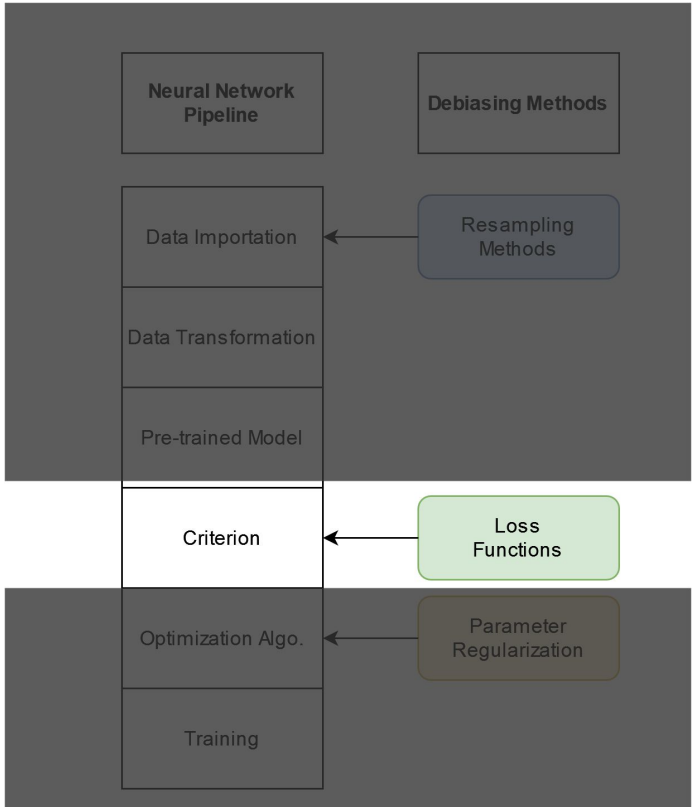
$$\mathrm{CB}_{\mathrm{sigmoid}}(\mathbf{z}, y) = -\frac{1-\beta}{1-\beta^{n_y}} \sum_{i=1}^{C} \log\left(\frac{1}{1+\exp(-z_i^t)}\right)$$

binary_cross_entropy_with_logits(raw_logits, labels)

$$\mathrm{CB}_{\mathrm{focal}}(\mathbf{z}, y) = -\frac{1-\beta}{1-\beta^{n_y}} \sum_{i=1}^{C} (1-p_i^t)^{\gamma} \log(p_i^t).$$

where γ is called the modulating factor

(ref)



Parameter Regularization

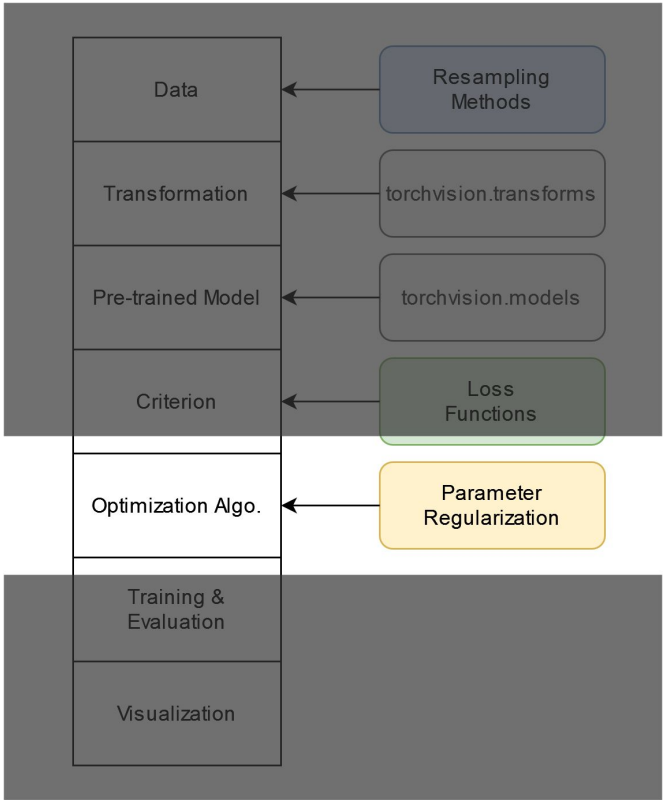
Weight balancing:

- A naive model has “artificially” large weights for common classes
- Try to balance the weights across classes for better predictions on the minority class

$$\Theta^* = \arg\min_{\Theta} \underbrace{F(\Theta; \mathcal{D})}_{\text{Neural network}} \equiv \sum_{i=1}^{N} \underbrace{\ell\big(f(\mathbf{x}_i; \Theta), y_i\big)}_{\text{Loss function}}$$

Optimized set of weights per layer of the network

(ref)



Parameter Regularization

$$\Theta^* = \arg\min_{\Theta} F(\Theta; \mathcal{D}) + \lambda \sum_k \|\theta_k\|_2^2,$$

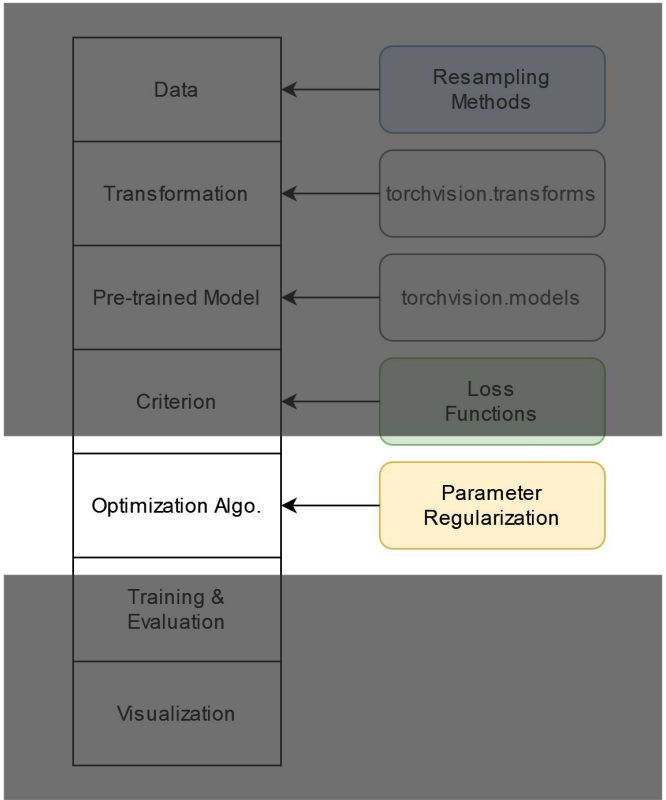
Hyperparameter

Weights for each class *k*

Weight decay limits weight growth and penalizes larger weights more severely than smaller weights, which promotes more balanced weight growth.

Weight decay decreases model complexity, which mitigates overfitting and improves generalization

(ref)



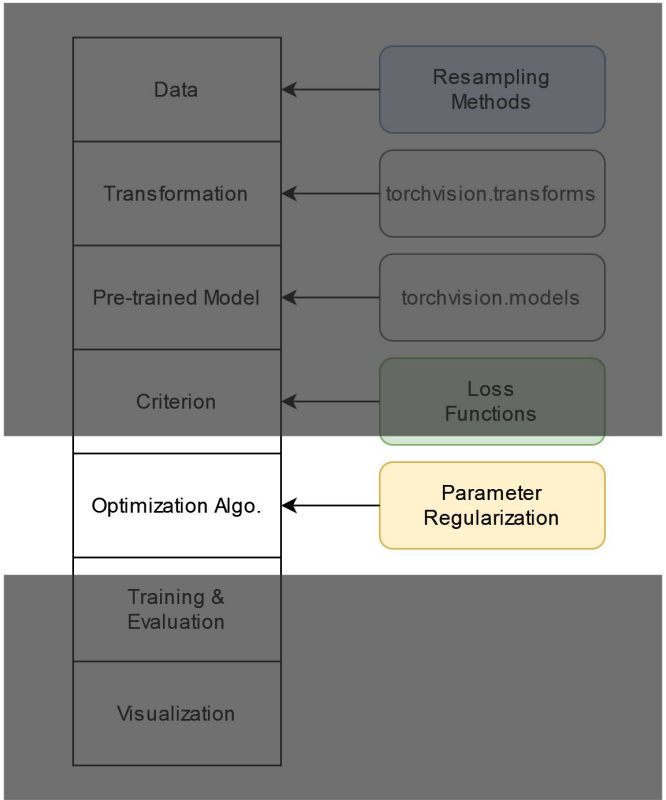
Parameter Regularization

L2- normalization perfectly balances classifier weights to be the unit norm

$$\mathbf{\Theta}^* = \arg\min_{\mathbf{\Theta}} F(\mathbf{\Theta}; \mathcal{D}), \quad s.t. \quad \|\boldsymbol{\theta}_k\|_2^2 = 1, \ \forall\, k.$$

Weights for each class *k*

Improves performance on the minority classes, but sacrifices common-class accuracy and overall model performance.



(ref)

Parameter Regularization

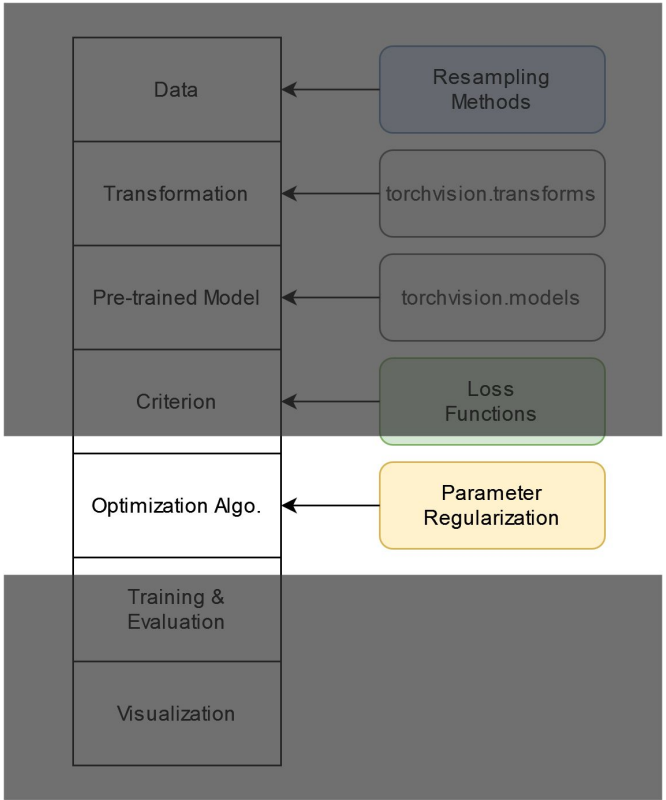
MaxNorm encourages smaller weight growth and places a limit on weight norms (δ)

$$\Theta^* = \arg\min_{\Theta} F(\Theta; \mathcal{D}), \quad s.t. \quad \|\theta_k\|_2^2 \leq \delta^2, \ \forall\, k,$$

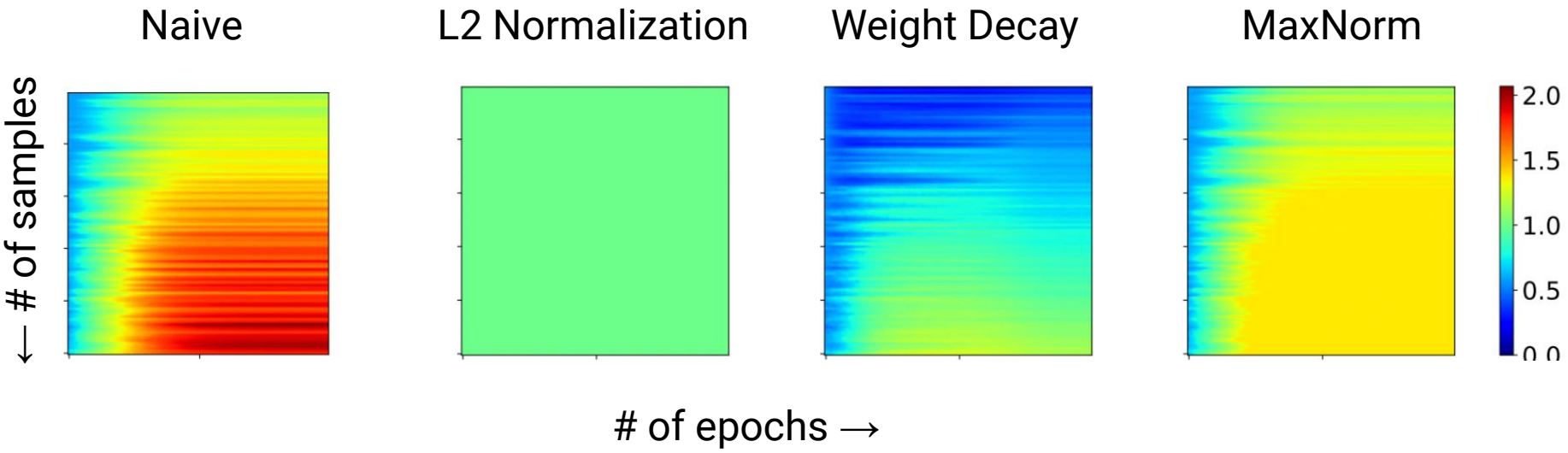
Weights for each class *k*

Weights cannot “explode” even at high learning rates because they are always bounded.

(ref)



Parameter Regularization



(Source)

Hyperparameter Optimization

Base Debiaser Functions

`transformer()`

- Image transformations
- Stored in `dict_transform`

`load_model()`

- Model selection
- Stored in `dict_model`

Parameters to Tune

```
params = {
        'optimizer__strategy': ['maxnorm'],
        'optimizer__thresh':[0.5, 2],
        'criterion__loss_family':["CB"],
        'criterion__beta': [0.5, 0.9],
        'criterion__loss_type': ["CE"]
        }
```

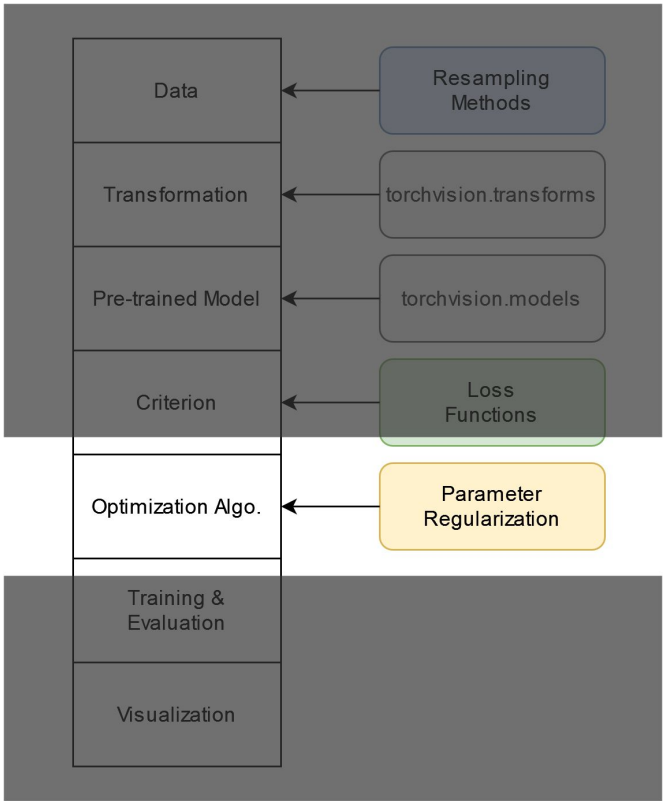
Main Tuning Function

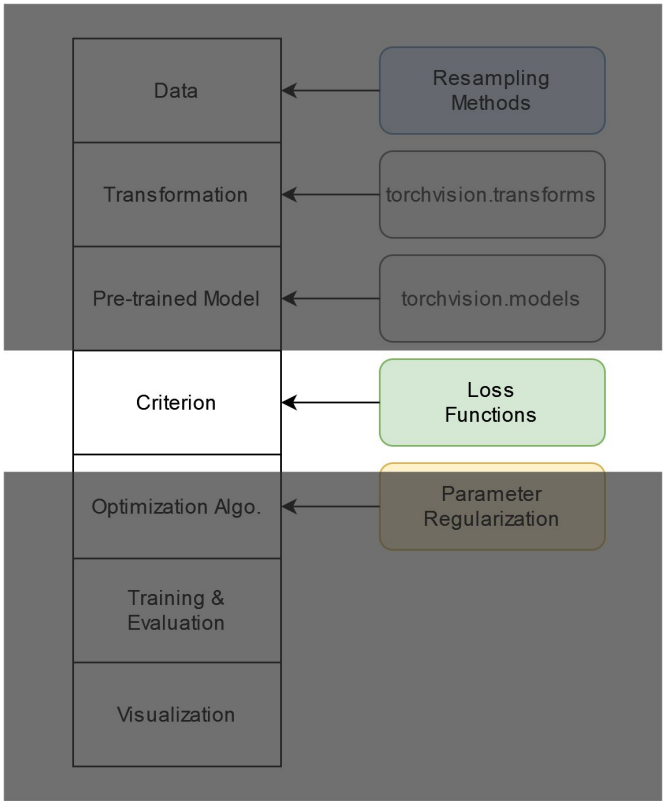
```
manual_opt():

    file_path,
    dict_transform,
    dict_model,
    params,
    best_by
    cv,
    sampling_strategy,
    random_state
```

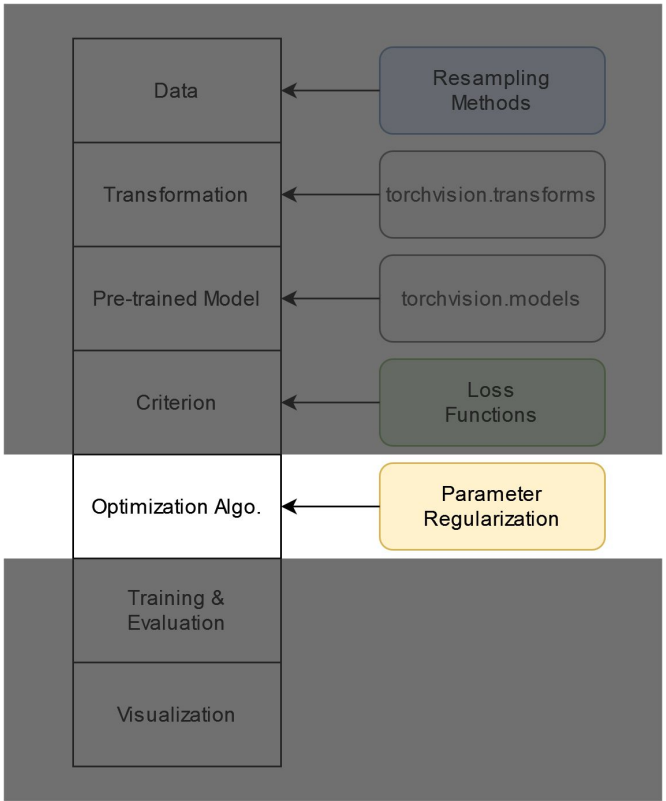
Hyperparameter Optimization

Output:

- **All** parameter combinations and their associated **mean validation accuracy,** and **mean precision and recall** for the minority class
- The metric used to determine the “best” cross-validation result
- The **best** parameter combination and its associated metric


Data
Resampling Methods
Transformation
torchvision.transforms
Pre-trained Model
torchvision.models
Criterion
Loss Functions
Optimization Algo.
Parameter Regularization
Training & Evaluation
Visualization


Data
Resampling Methods
Transformation
torchvision.transforms
Pre-trained Model
torchvision.models
Criterion
Loss Functions
Optimization Algo.
Parameter Regularization
Training & Evaluation
Visualization


Data
Resampling Methods
Transformation
torchvision.transforms
Pre-trained Model
torchvision.models
Criterion
Loss Functions
Optimization Algo.
Parameter Regularization
Training & Evaluation
Visualization