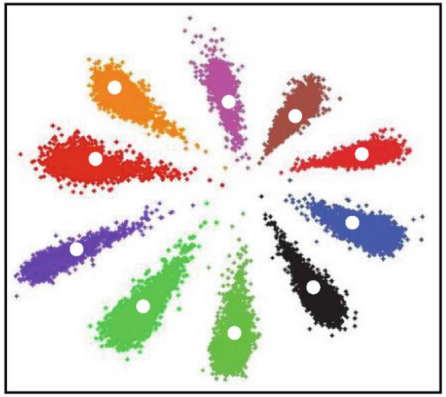
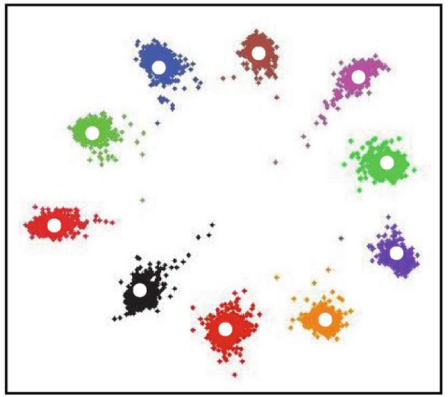
ANN: *de novo* training…

select a loss function appropriate to the data and task

- (multi–)regression: Mean Squared Error (MSE)
  - the ‘default’
  - assumes Gaussian distribution
- (multi–)regression: Mean Absolute Error (MAE)
  - more tolerant of outliers than MSE
- multi–regression: cosine similarity
  - mean of squared error between true and predicted



(a) $\lambda = 0.001$



(c) $\lambda = 0.1$

ANN: …*de novo* training…

(multi–label) classification: Cross–Entropy (CE)

- the ‘default’
- difference between correct/incorrect classification
- comes in may ‘flavors’ for data/output type

classification: hinge

- maximizes margin between correct/incorrect classification

(multi–label) classification: Kullback–Leibler (KL) divergence

- relative difference between correct/incorrect classification
- non–metric: sometimes causing training instability

(Wen et al. 2016; https://doi.org/10.1007/978-3-319-46478-731)

ANN: …*de novo* training…

translation: Kullback–Leibler divergence

translation: Mean Squared Error (MSE)

translation: cosine similarity

translation: Dice

- measures similarity between two samples
- sometimes unstable in training

ANN: …*de novo* training…

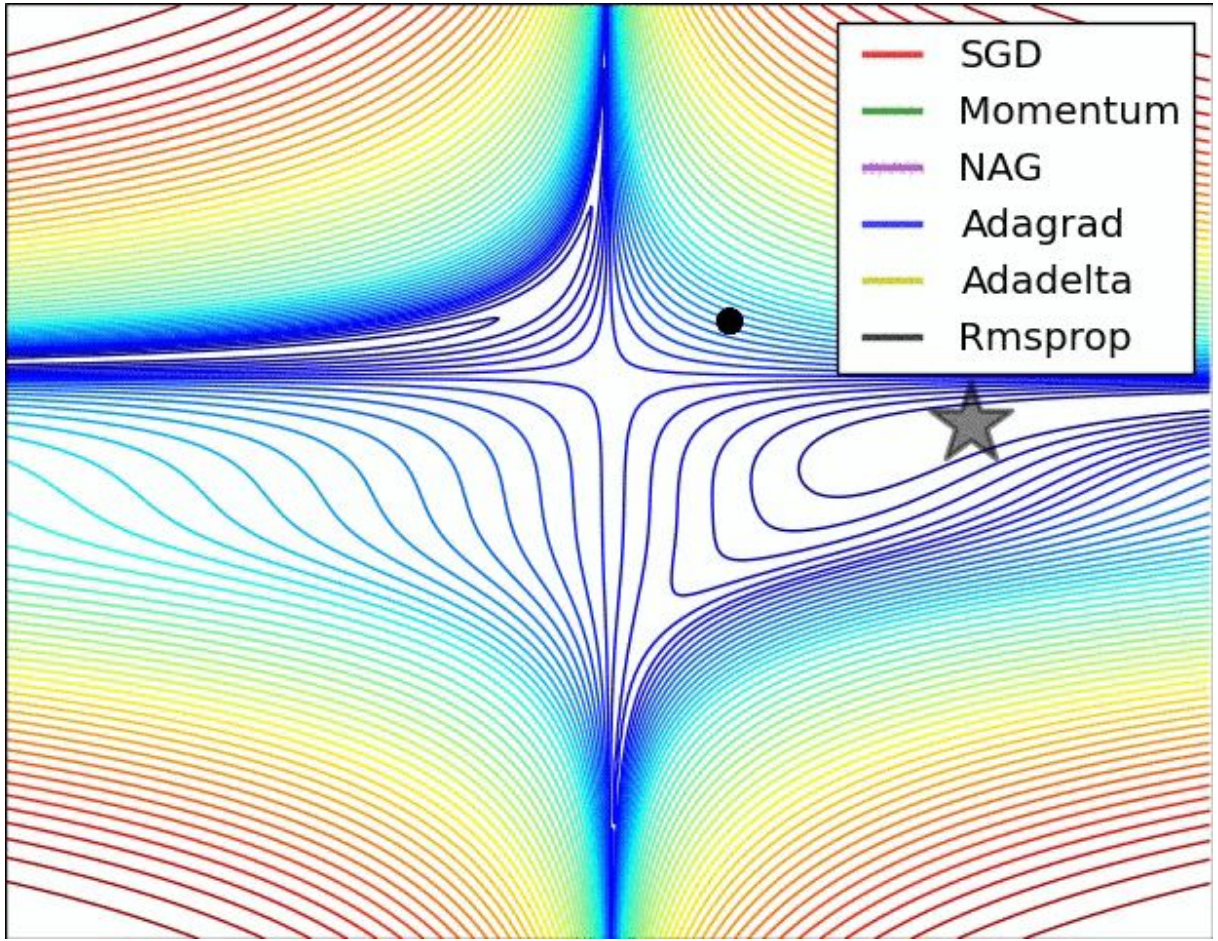
select metrics to measure model performance

- appropriate to the data and task
- loss functions and their modifications
- regression: r^2
- classification: accuracy, precision, recall, AUC, F_1
- translation: similarity, intersection over union (IoU)

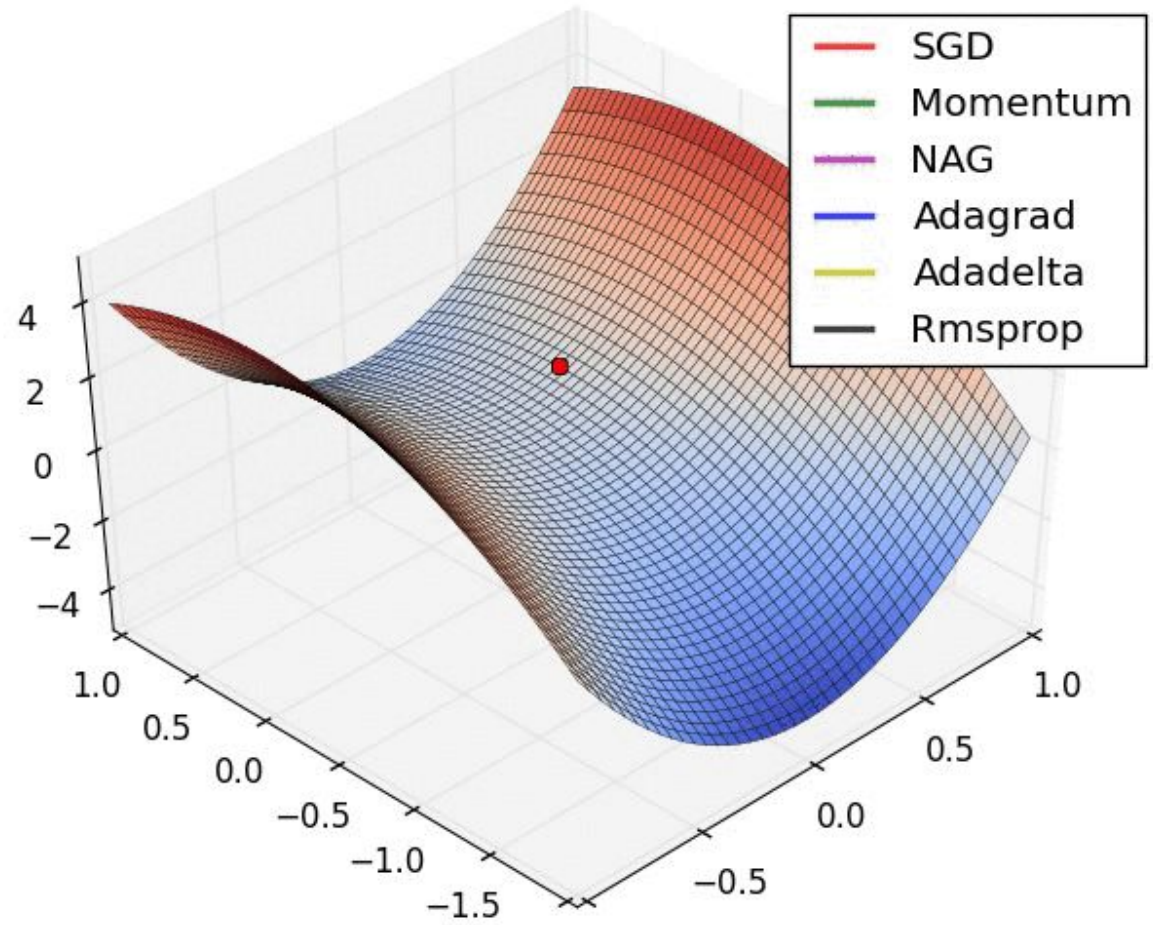
ANN: …*de novo* training…

select an optimizer matching model architecture and loss

- updates model for each sample
  - distributes error to layers using Back Propagation
- Stochastic Gradient Descent (SGD)
  - will not converge unless learning rate is ‘scheduled’
- Adaptive Moment Estimation (Adam)
  - has a (decaying) memory of past average gradients
  - also benefits from learning rate scheduling



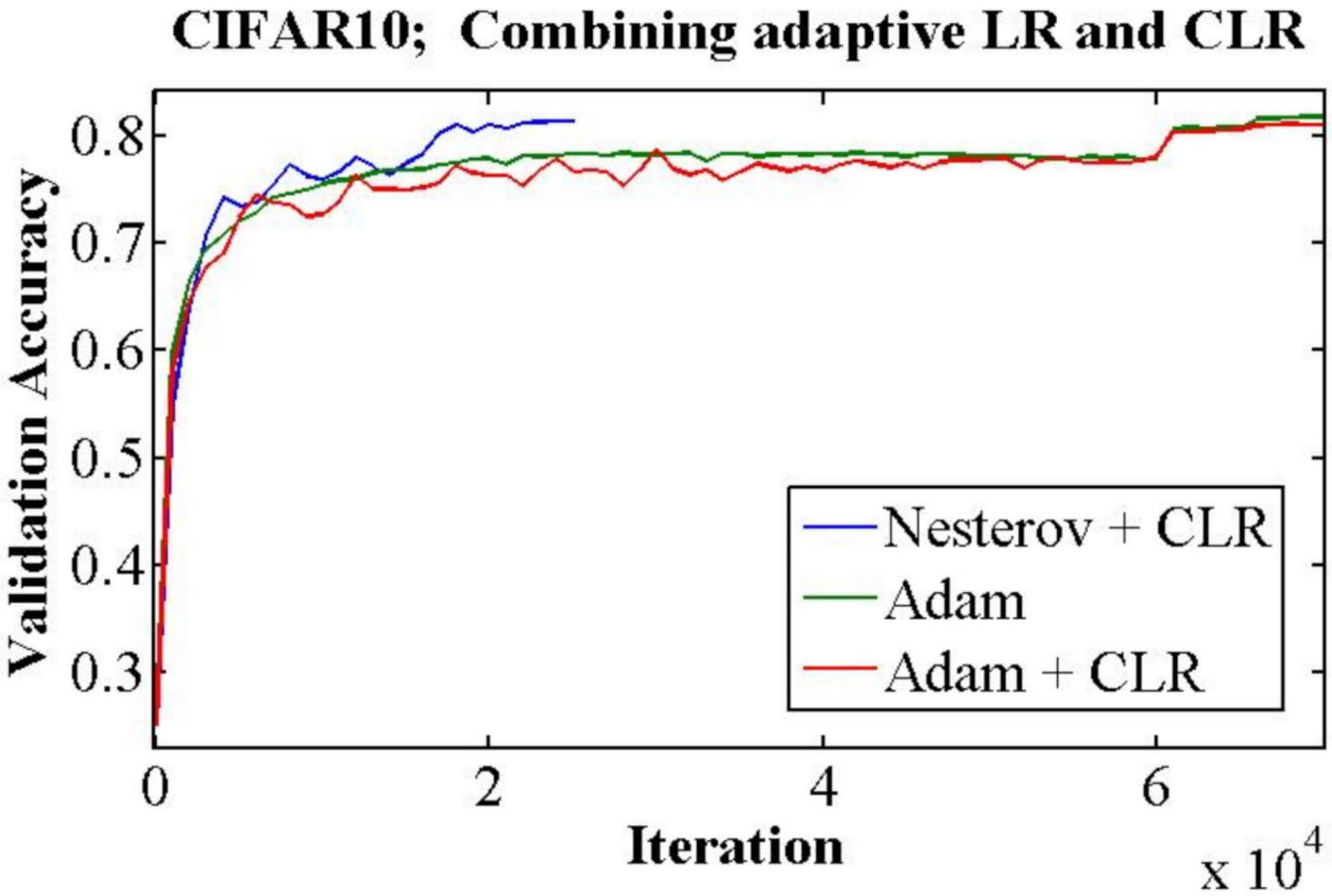
https://www.ruder.io/optimizing-gradient-descent/



https://www.ruder.io/optimizing-gradient-descent/

ANN: …*de novo* training…

- stop training while you are ahead: monitor a metric, stop when progress ceases
  - “Early stopping (is) beautiful free lunch” (G. Hinton)
- select a learning rate scheduler
  - formulae based on epoch number
  - Cyclical Learning Rates (CLR)
    - can use a high rate while maintaining training stability
    - must select an upper bound
      - dataset, training parameter, and model dependent
      - select by trial and error



Cyclical Learning Rates (Smith 2015; https://arxiv.org/abs/1506.01186)

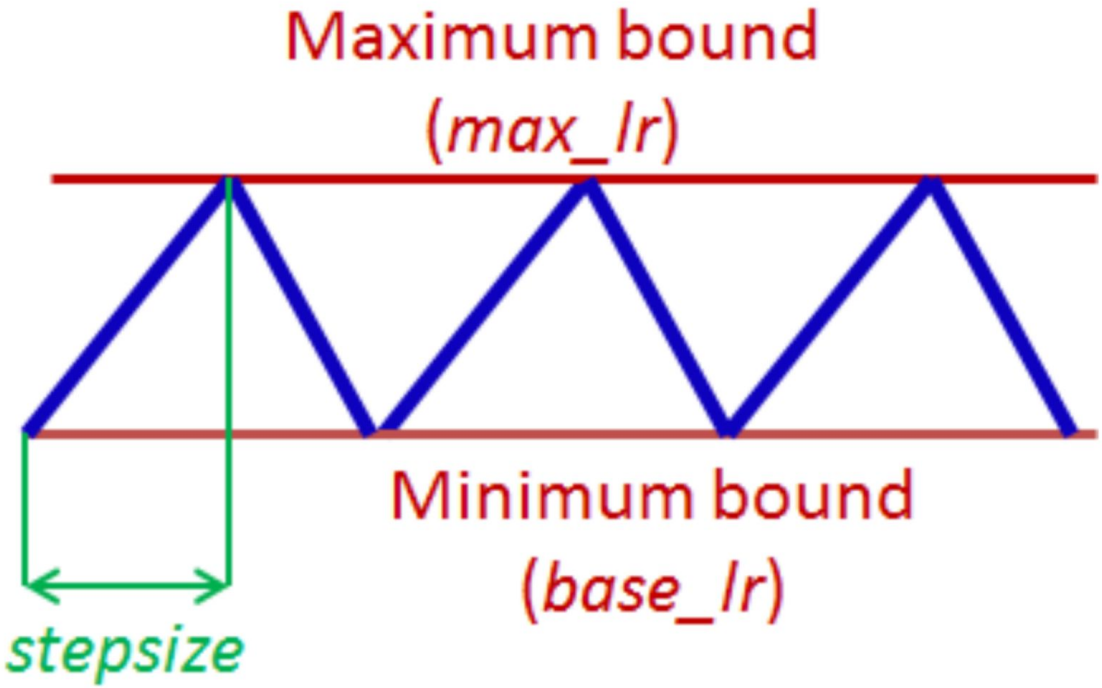


Figure 2. Triangular learning rate policy. The blue lines represent learning rate values changing between bounds. The input parameter $stepsize$ is the number of iterations in half a cycle.

Cyclical Learning Rates (Smith 2015; https://arxiv.org/abs/1506.01186)

ANN: …*de novo* training…

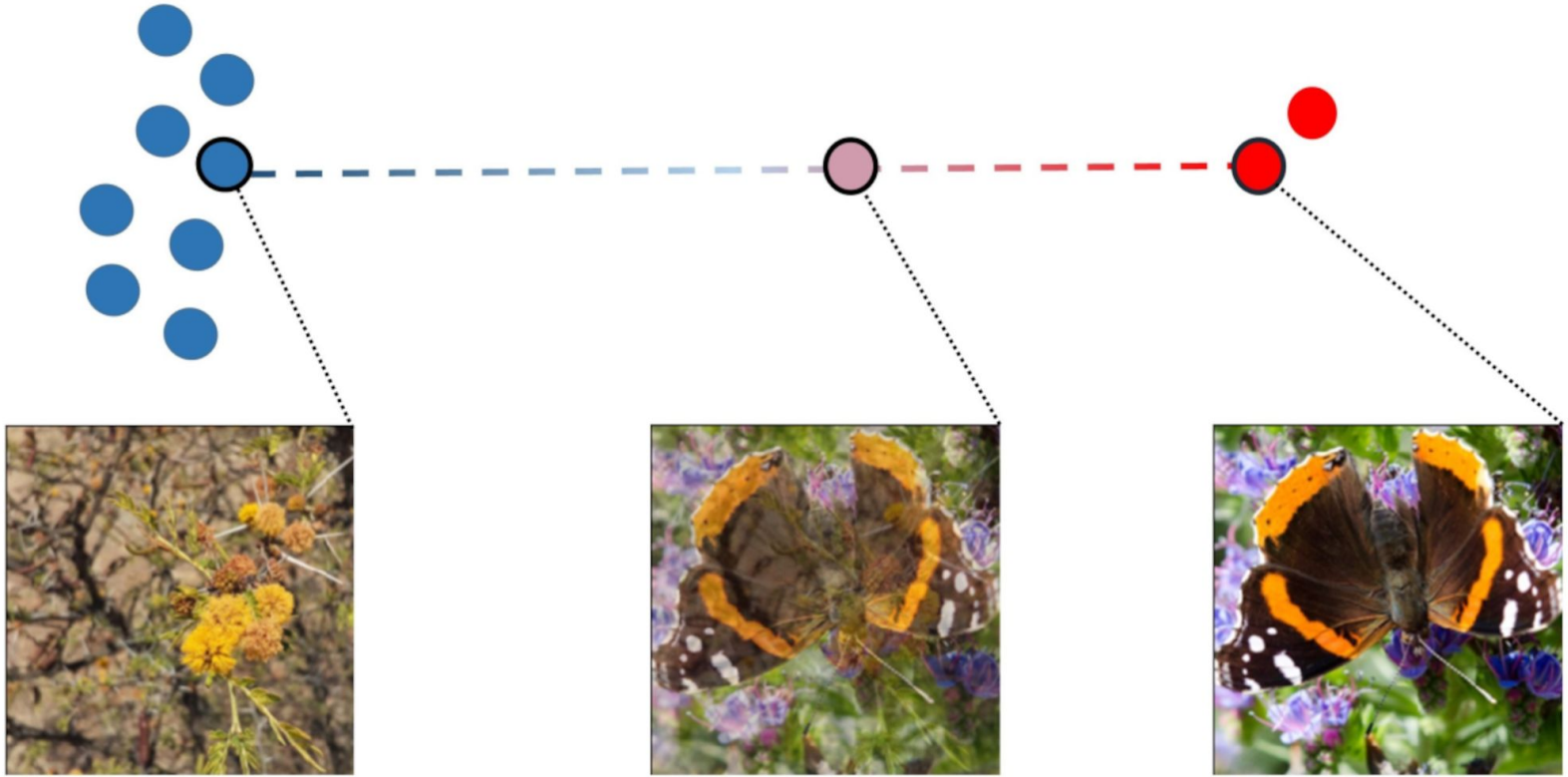
‘balance’ dataset by resampling (downsample is better)

or use a loss function that is insensitive to imbalance

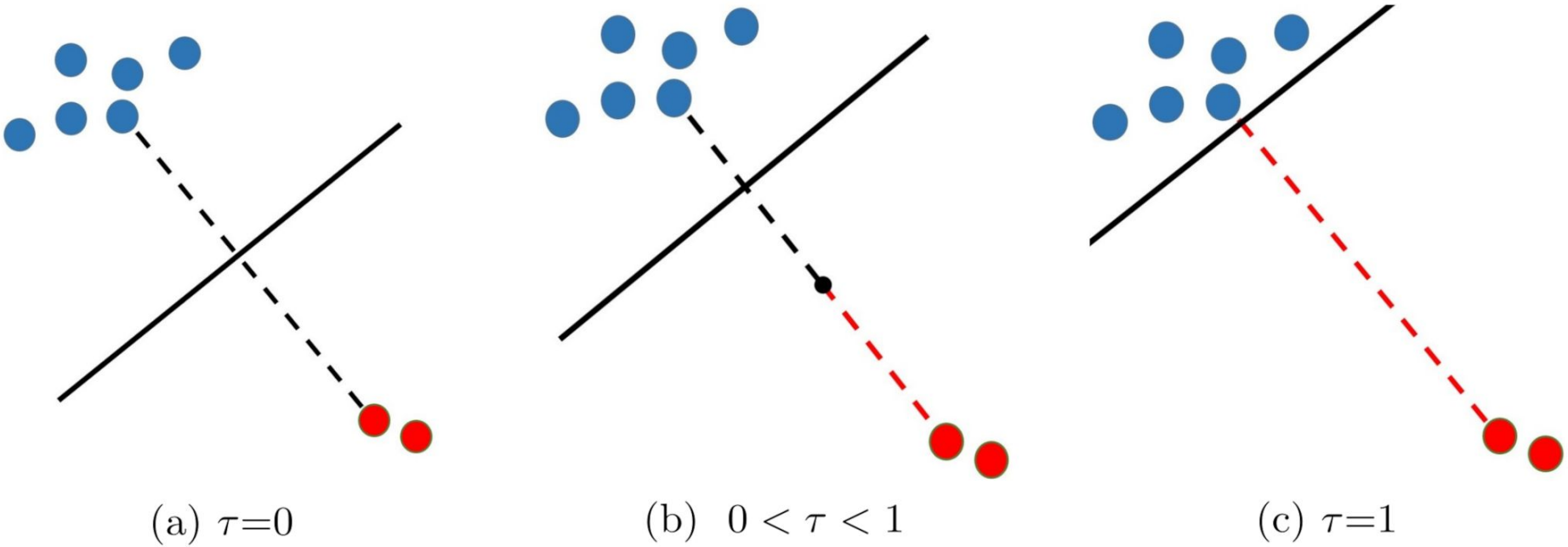
or augment data to (somewhat) ameliorate bias (e.g. ReMix)

or select weights to (somewhat) ameliorate bias

- inverse class frequency:
  - the ‘default’
  - can overweight high–frequency classes
- effective class size:
  - weights based on sample information content



ReMix (Chou et al. 2020; https://arxiv.org/abs/2007.03943)



ReMix (Chou et al. 2020; https://arxiv.org/abs/2007.03943)

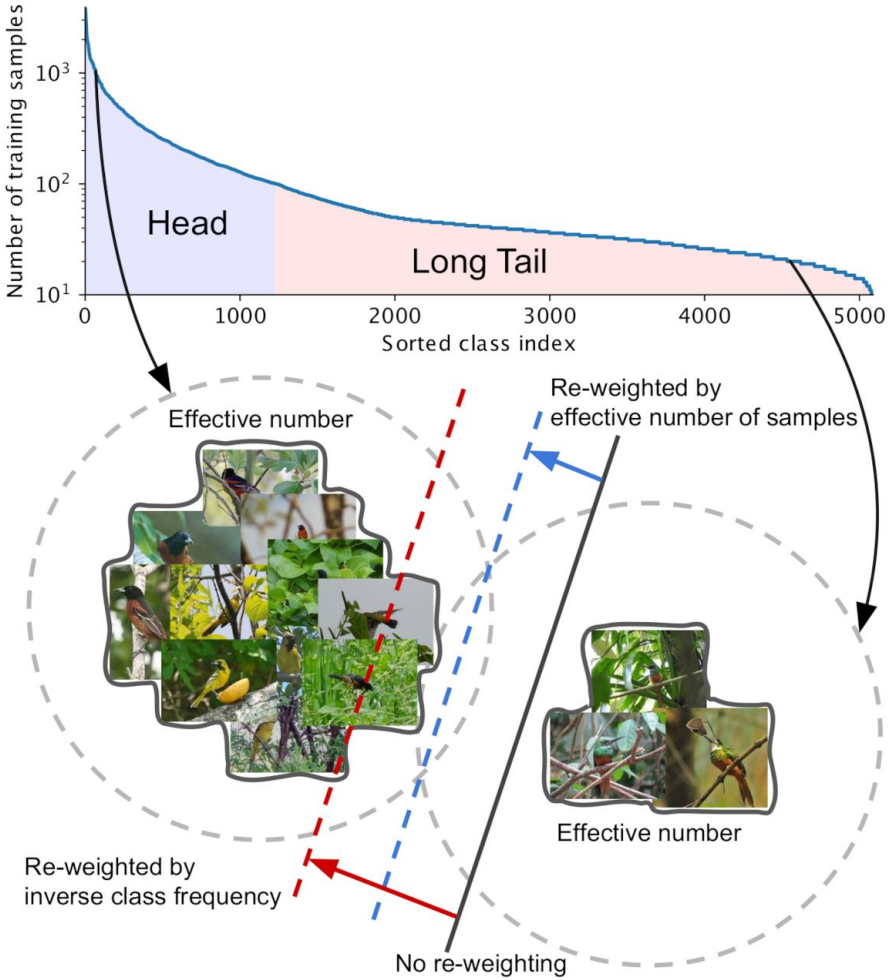


Figure 1. Two classes, one from the head and one from the tail of a long-tailed dataset (iNaturalist 2017 [40] in this example), have drastically different number of samples. Models trained on these samples are biased toward dominant classes (black solid line). Re-weighing the loss by inverse class frequency usually yields poor performance (red dashed line) on real-world data with high class imbalance. We propose a theoretical framework to quantify the effective number of samples by taking data overlap into consideration. A class-balanced term is designed to re-weight the loss by inverse effective number of samples. We show in experiments that the performance of a model can be improved when trained with the proposed class-balanced loss (blue dashed line).

(Cui et al. 2019; https://arxiv.org/abs/1901.05555)

Dataset Name	Long-Tailed CIFAR-10						Long-Tailed CIFAR-100					
Imbalance	200	100	50	20	10	1	200	100	50	20	10	1
Softmax	**34.32**	29.64	25.19	17.77	13.61	6.61	65.16	61.68	56.15	48.86	44.29	29.07
Sigmoid	34.51	**29.55**	23.84	**16.40**	**12.97**	**6.36**	64.39	**61.22**	55.85	48.57	44.73	**28.39**
Focal ($\gamma = 0.5$)	36.00	29.77	**23.28**	17.11	13.19	6.75	65.00	61.31	55.88	48.90	44.30	28.55
Focal ($\gamma = 1.0$)	34.71	29.62	23.29	17.24	13.34	6.60	**64.38**	61.59	**55.68**	**48.05**	**44.22**	28.85
Focal ($\gamma = 2.0$)	35.12	30.41	23.48	16.77	13.68	6.61	65.25	61.61	56.30	48.98	45.00	28.52
Class-Balanced	**31.11**	**25.43**	**20.73**	**15.64**	**12.51**	**6.36***	**63.77**	**60.40**	**54.68**	**47.41**	**42.01**	**28.39***
Loss Type	SM	Focal	Focal	SM	SGM	SGM	Focal	Focal	SGM	Focal	Focal	SGM
β	0.9999	0.9999	0.9999	0.9999	0.9999	-	0.9	0.9	0.99	0.99	0.999	-
γ	-	1.0	2.0	-	-	-	1.0	1.0	-	0.5	0.5	-

Table 2. Classification error rate of ResNet-32 trained with different loss functions on long-tailed CIFAR-10 and CIFAR-100. We show best results of class-balanced loss with best hyperparameters (SM represents Softmax and SGM represents Sigmoid) chosen via cross-validation. Class-balanced loss is able to achieve significant performance gains. $*$ denotes the case when each class has same number of samples, class-balanced term is always 1 therefore it reduces to the original loss function.

(Cui et al. 2019; https://arxiv.org/abs/1901.05555)

ANN: …*de novo* training…

augment training data

- often required to prevent overfitting
- algorithmically randomly modifies training data
  - change size, shape, rotation, colors, etc.
- mixup is often best for color or sequence augmentation
- zoom and shear are often best for ‘shape’ augmentation
- within batch implementations, can be used on GPU/TPU

ResNet-50 Mixup [46] Cutout [2] CutMix

(Yun et al. 2019; https://arxiv.org/abs/1905.04899)

ANN: …*de novo* training

DNA/AA augmentation

- ‘mutate’ using a substitution matrix (e.g. BLOSUM62)
- ‘mutate’ using a substitution model
- ‘mutate’ using a reduced AA alphabet
- ‘mutate’ using an alignment profile
- ‘mutate’ DNA using synonymous codons

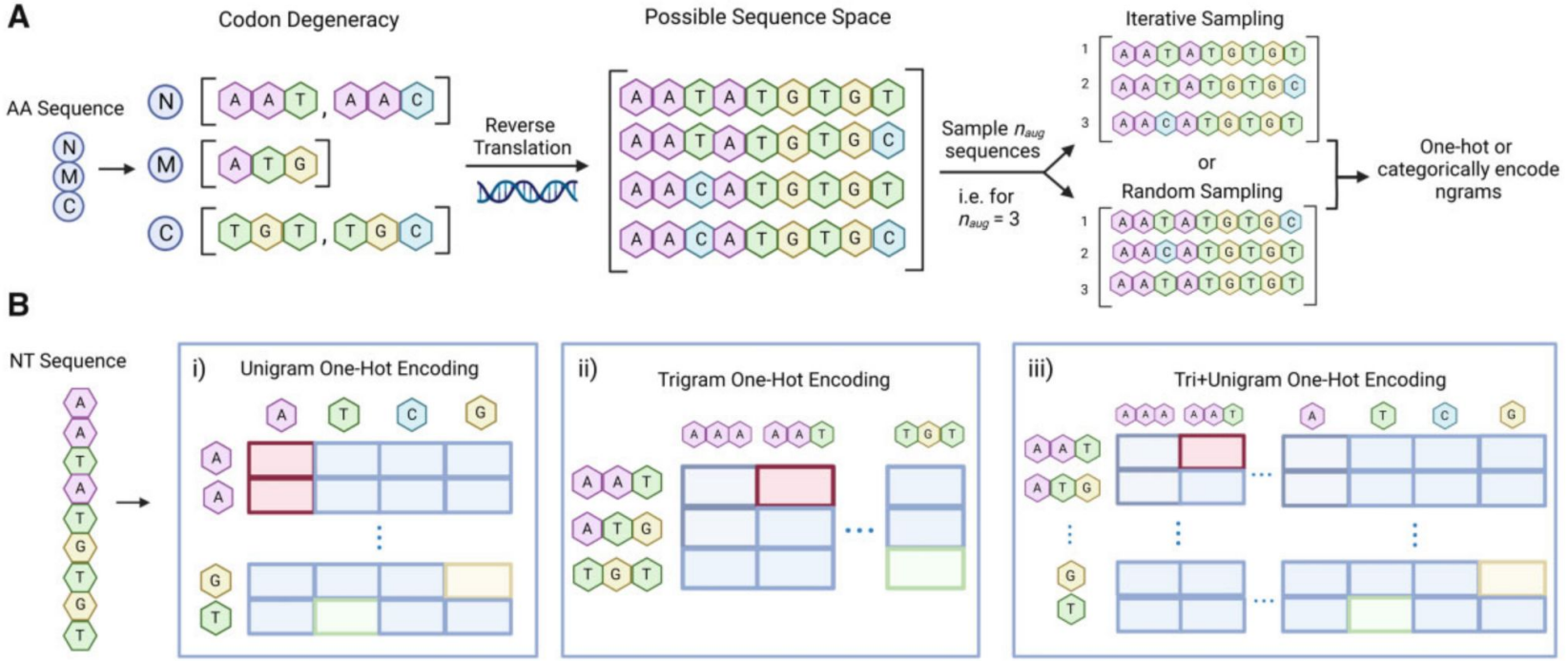
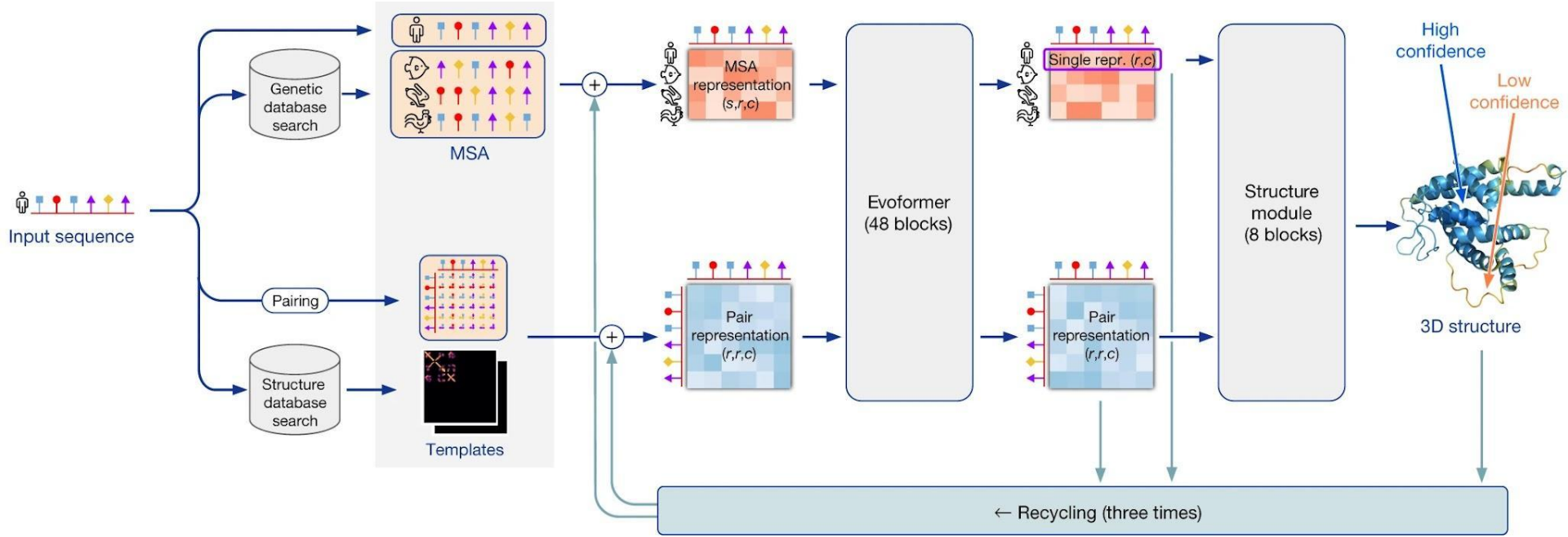


Fig. 1. (**A**) Nucleotide augmentation (NTA) approach. Possible nucleotide codons are determined for each residue in an input amino acid sequence. Full-length nucleotide sequences are then sampled n_{aug} times from the possible sequence space via iterative or random sampling, where n_{aug} is a user-specified augmentation factor. The sequences are then either categorically encoded (Transformer) or one-hot encoded (CNN). The different ngram one-hot encoding approaches are illustrated in (**B**). First, a nucleotide sequence is broken into (B_i) unigrams (B_{ii}) trigrams or (B_{iii}) trigrams concatenated with unigrams (tri+unigrams). The ngrams are then tokenized according to their respective vocabularies. The tokenized vector corresponds to the Transformer model's input; however, this vector undergoes an additional one-hot encoding step for the CNN. For example, the resulting one-hot matrix size for a nucleotide sequence of length 9 will either be 9×4 (unigrams B_i), 3×64 (trigrams, B_{ii}) or 12×68 (tri+unigrams B_{iii}). Note that amino acids are encoded as unigrams i.e. for sequence length L the resulting one-hot matrix is L×20 for the 20 canonical amino acids. Created with Biorender.com

NTA (Minot and Reddy 2022; https://doi.org/10.1093/bioadv/vbac094)



AlphaFold (Jumper et al. 2021; https://doi.org/10.1038/s41586-021-03819-2)

ANN: transfer learning

makes training (much) faster

makes data imbalance less problematic

use a model trained on another (diverse, balanced) dataset

pretrain a model with (similar) data, retrain with your data

[0] replace the output layer with a randomly initialized one

[1] freeze all other layers

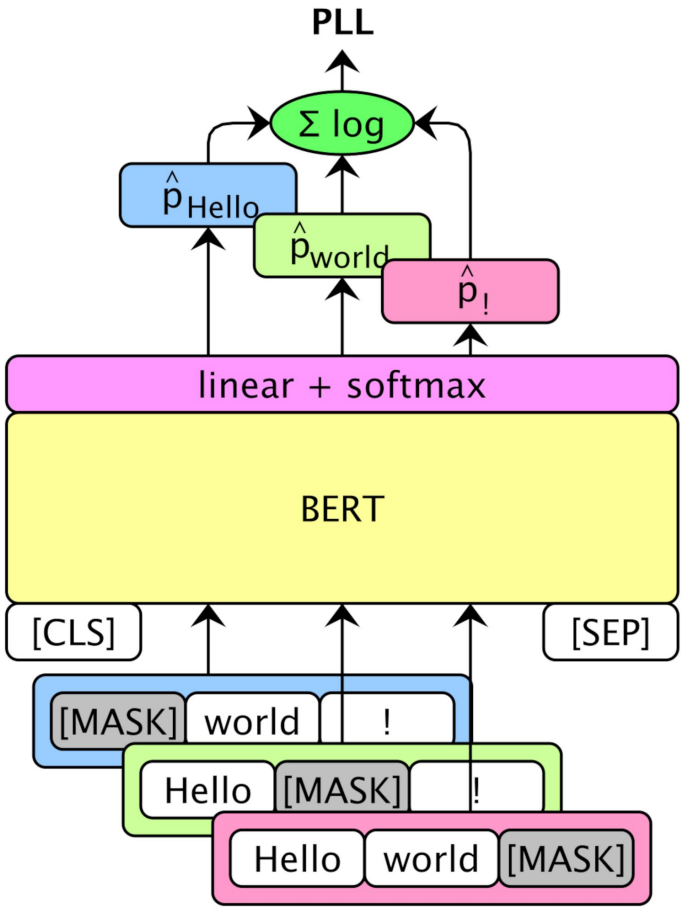
[2] train model on new data

[3] slowly unfreeze one layer at a time and continue train

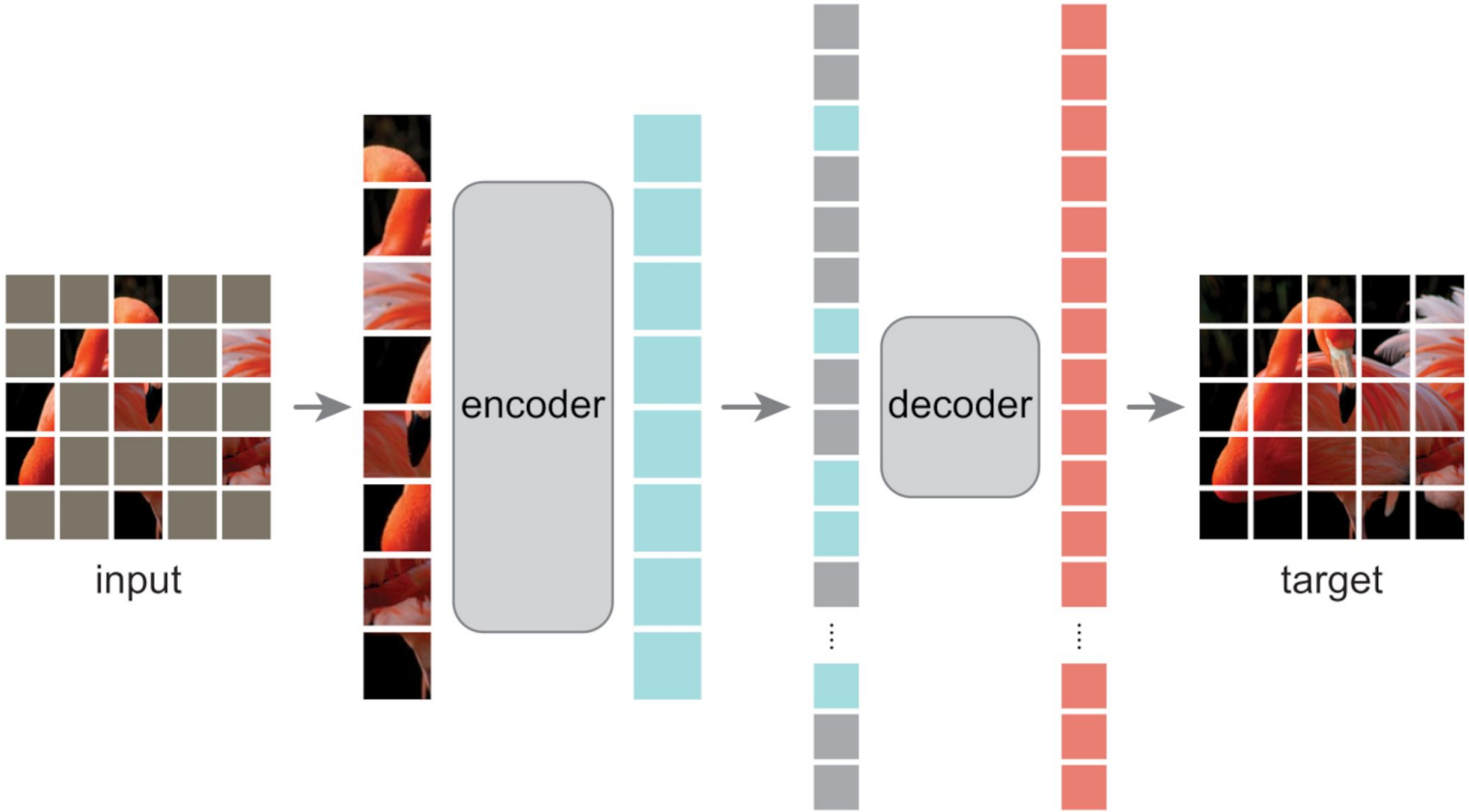
ANN: self–supervised pretraining

pretrain a model using ‘unlabeled’ data then transfer learning

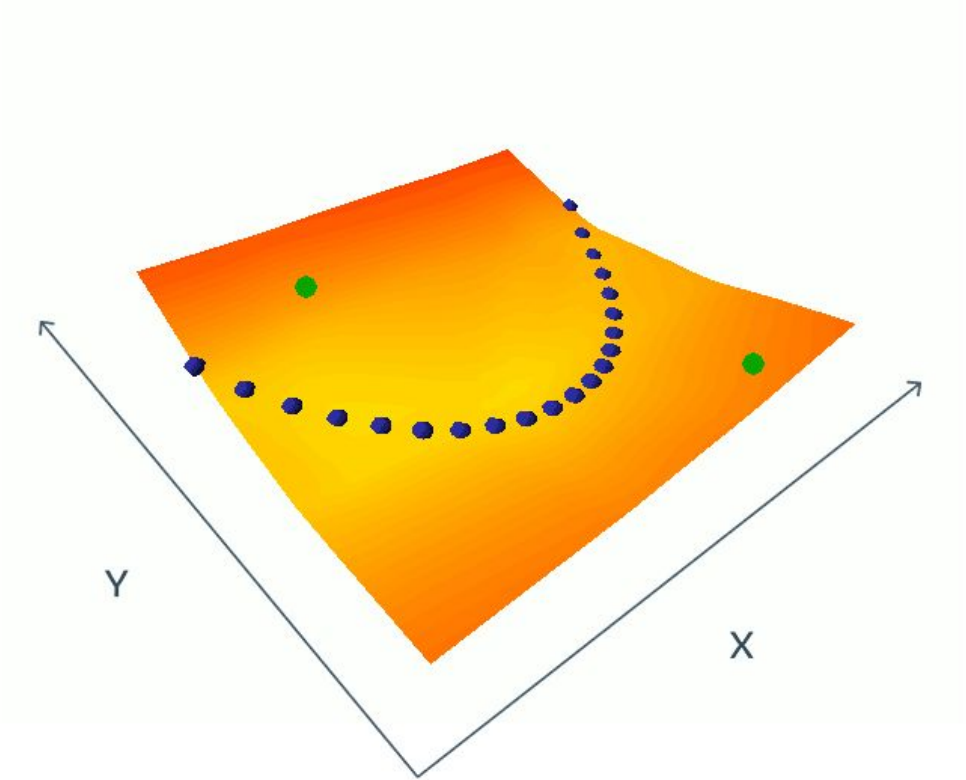
- abundant labels for another (related) task
  - e.g. meta data from standard datasets
- labels that can be calculated with a known algorithm
- create a pretext (proxy) task from existing data
  - obscure data, model learns to recreate the original



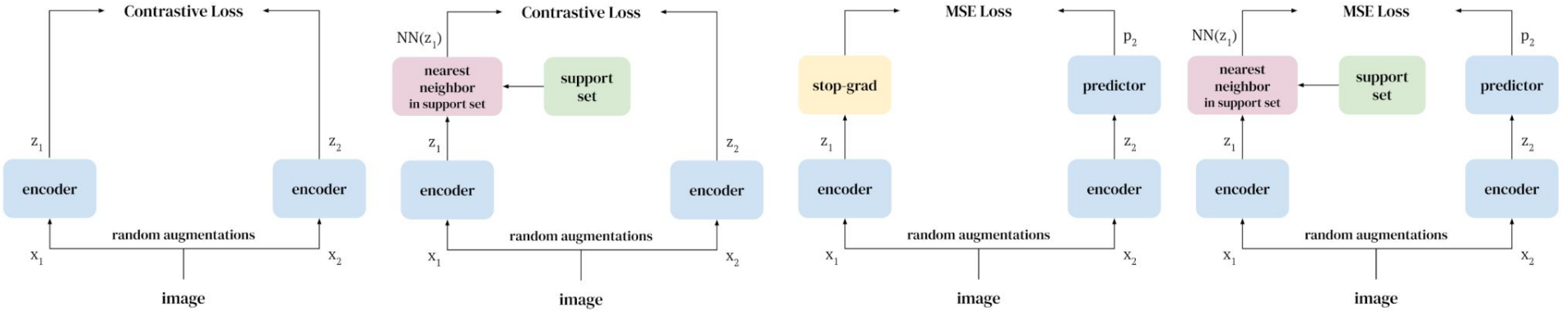
masked language model training (Salazar et al. 2021; https://arxiv.org/abs/1910.14659)



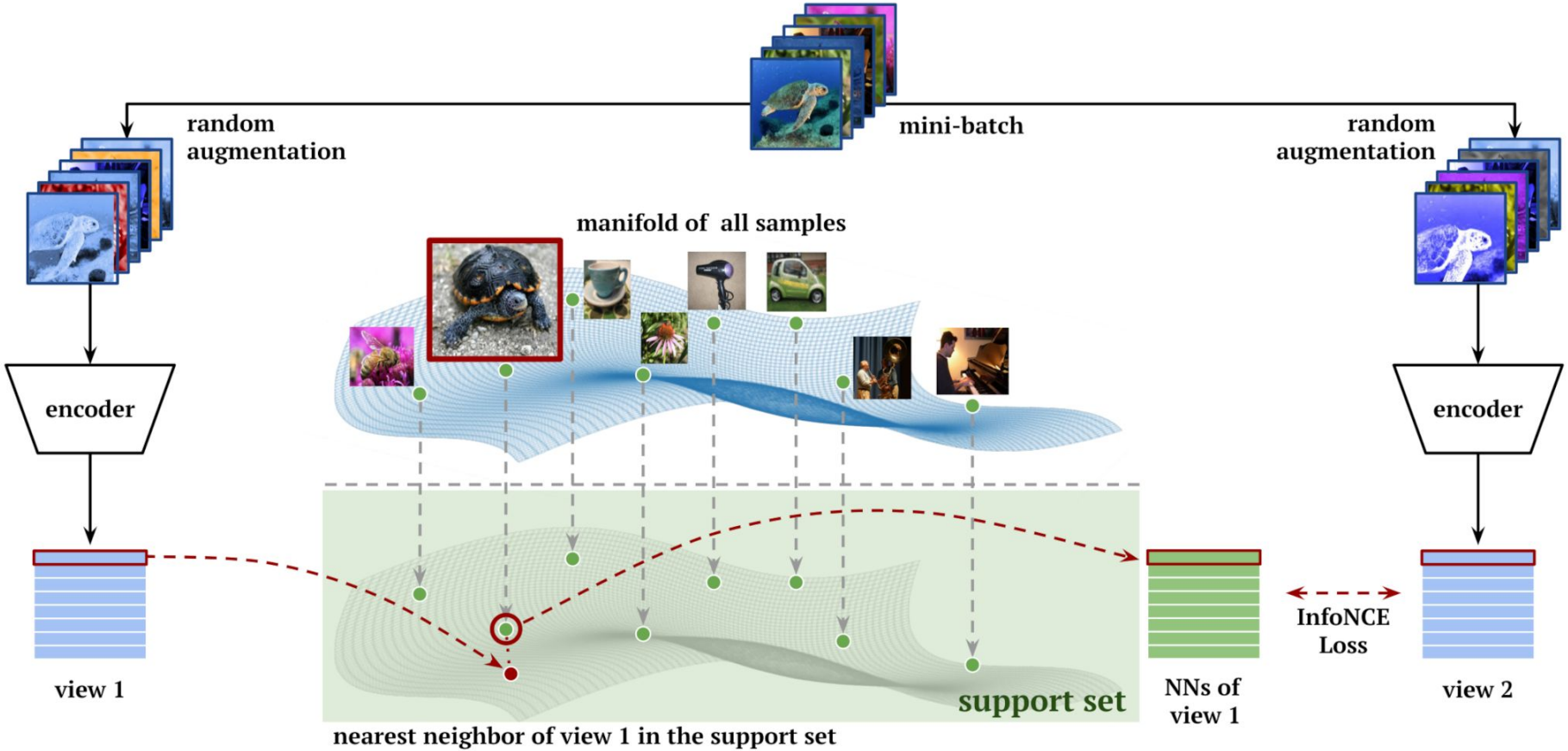
Masked Autoencoders (MAE; He et al. 2021; https://arxiv.org/abs/2111.06377)



Contrastive SSL (https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/)



Nearest-Neighbor Contrastive Learning (Dwibedi et al. 2021; https://arxiv.org/abs/2104.14548)



Nearest-Neighbor Contrastive Learning (Dwibedi et al. 2021; https://arxiv.org/abs/2104.14548)

ANN: distillation

convert a large ensemble of models into one small model

faster (fewer calculations) and more compact for deployment

output the raw logit values for the large model(s)

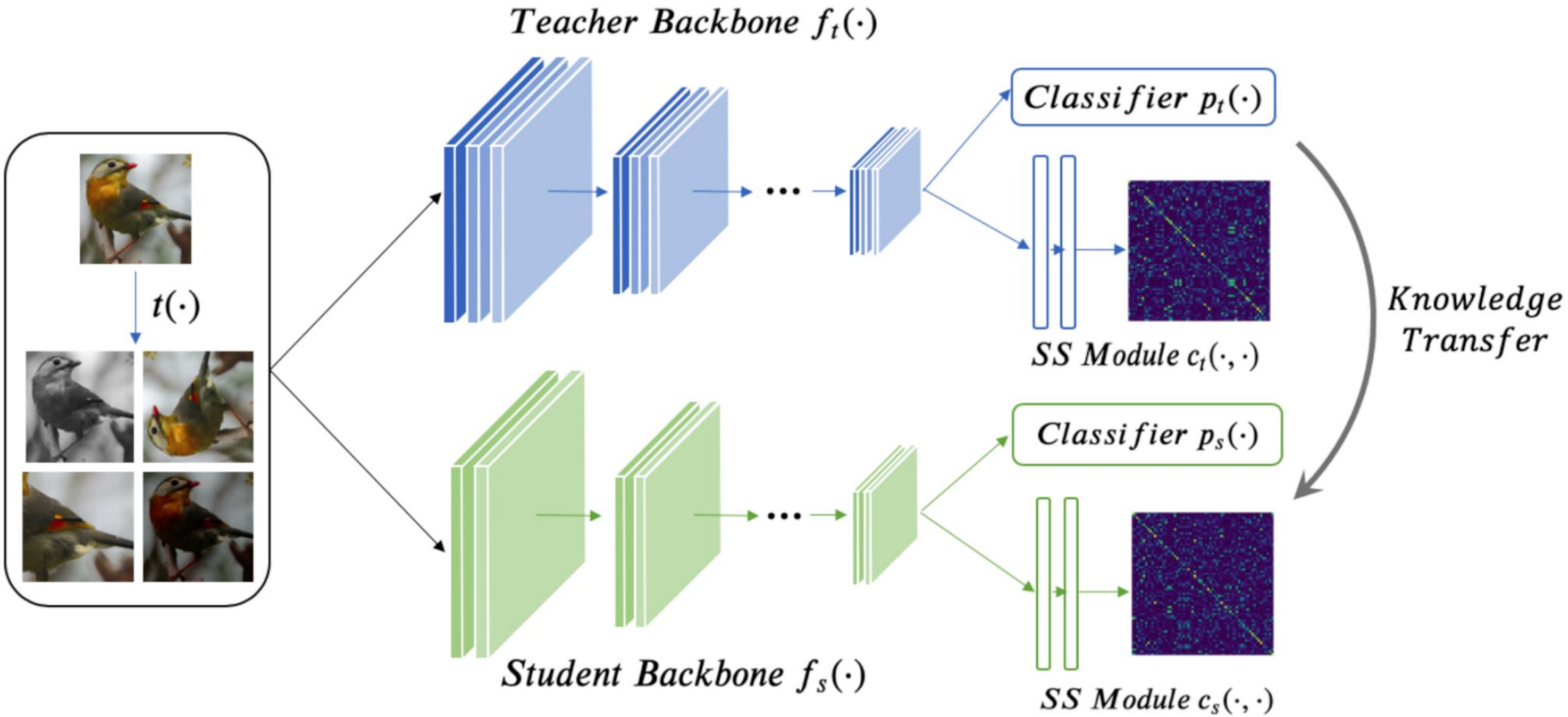
- preserves samples difficulty information
- makes edge cases easier to classify

train the small model to output the logit values (KL divergence)

- important that the values are for the same augmented input

add an activation layer to the small model

- generally performs about as well as the large model(s)



self–supervised knowledge distillation (Xu et al. 2020; https://arxiv.org/abs/2006.07114)

ANN: framework sampler

TensorFlow:	an open source Alphabet (Google) product
	a front end (Python, JavaScript, etc.) and C++ backend
	can deploy ANN to edge devices easily
	TensorFlow Lite, TensorFlow.js
PyTorch:	an open source Meta (FaceBook) product
	a front end (Python, C++) and C++ backend
ONNX:	an open source Microsoft product
	an open interchange format and inference platform
	can deploy ANN to edge devices ‘easily’

ANN: how to—dataset

define a clear and simple question/answer

- every complication adds additional training difficulty

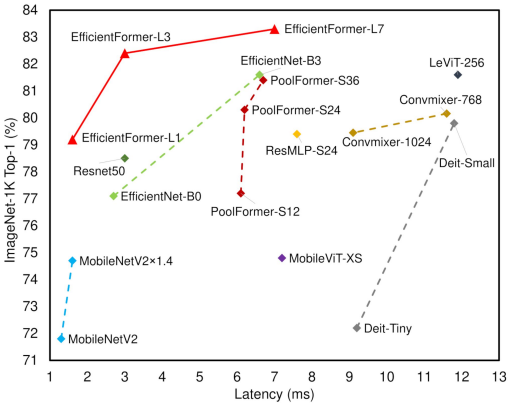
strive for big, balanced, high–quality labels, no missing data

- uniform data generation and labeling
- pay attention to data licensing

make a tiny representative microcosm for development

- use tiny images (e.g. 32^2 px), fewer samples
- what works for a microcosm generally works for a larger dataset
- scale to the full dataset after architecture and training are fixed

ANN: how to—model



match architecture type to available hardware and dataset

- look for ‘edge’, ‘embedded’, ‘mobile’, ‘lightweight’ architectures
- test on your data or similar benchmark datasets

start with existing scalable architectures

- develop using a tiny model that can scale up (if needed)
- use predefined scalings (or customize for your dataset)

add other architectures as needed

- ensemble or just the one with the best performance

aim to iterate very quickly (ca. 5 minutes per ‘good’ microcosm training)

(Li et al. 2022; https://arxiv.org/abs/2206.01191)

ANN: how to—*de novo* training

aim to iterate very quickly (ca. 5 minutes on microcosm)

figure out how to avoid weights

- downsampling or augmentation (e.g. ReMix)

pick a loss function matching data and model architecture

for each loss function, optimize batch and learning rate

use the largest batch that hardware and model will allow

- optimize learning rate after batch size

find a good augmentation for data and model architecture

- possibly modify model to prevent overfitting (dropout, noise injection)

ANN: how to—transfer learning

de novo training advise plus:

find high–quality, balanced, large, proxy datasets

- similar problem domain (e.g. image classification)
- (much) more complex than your dataset
- evaluate transferability for each proxy dataset

train model on proxy to ‘saturation’ then transfer