DeepMind

Learning Optimal Conformal Classifiers


David Stutz


Krishnamurthy (Dj) Dvijotham


Ali Taylan Cemgil

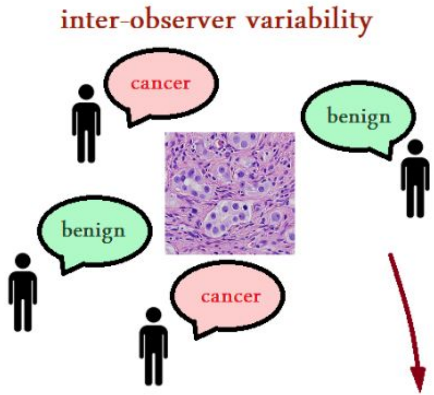

Arnaud Doucet

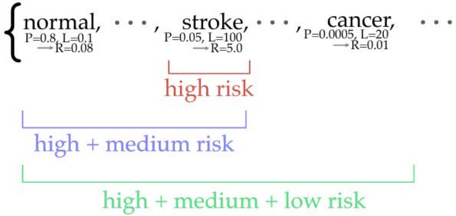
ICLR 2022

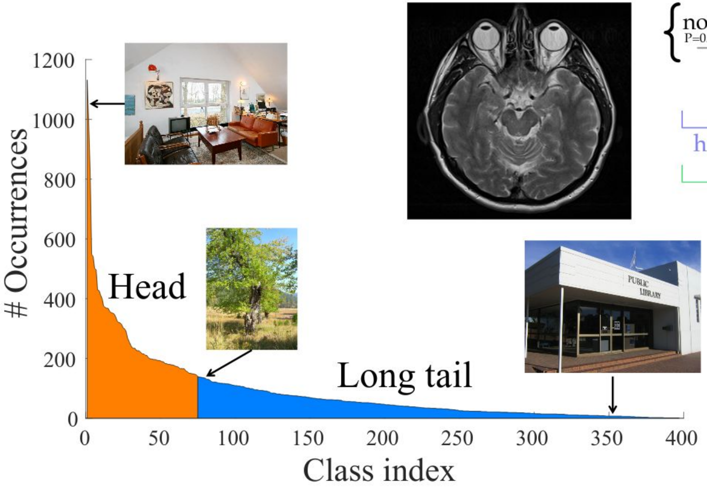
Ambiguity in AI

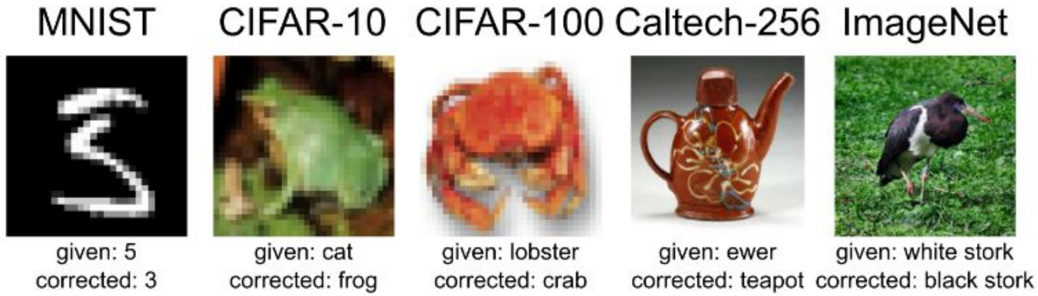
Public

- True ground truth unknown / label errors
- Rare classes or long-tailed class distribution
- High-stakes and security-critical applications





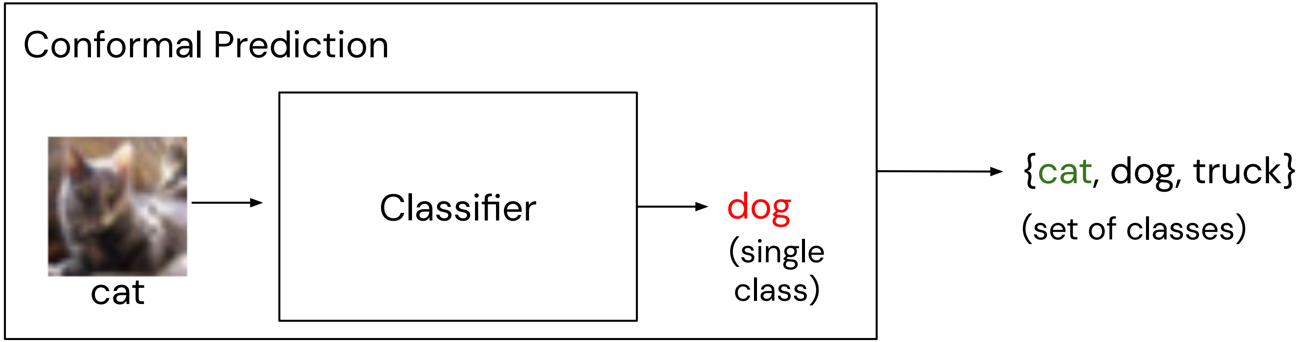




Wang et al. Learning to Model the Tail, 2017; Karimi et al., Deep learning with noisy labels: exploring techniques and remedies in medical image analysis, 2020; Bates et al., Distribution-Free, Risk-Controlling Prediction Sets, 2021; Northcutt et al., Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks, 2021.

Overview and Motivation: Conformal Prediction

Split conformal prediction as post-training wrapper with coverage guarantee:

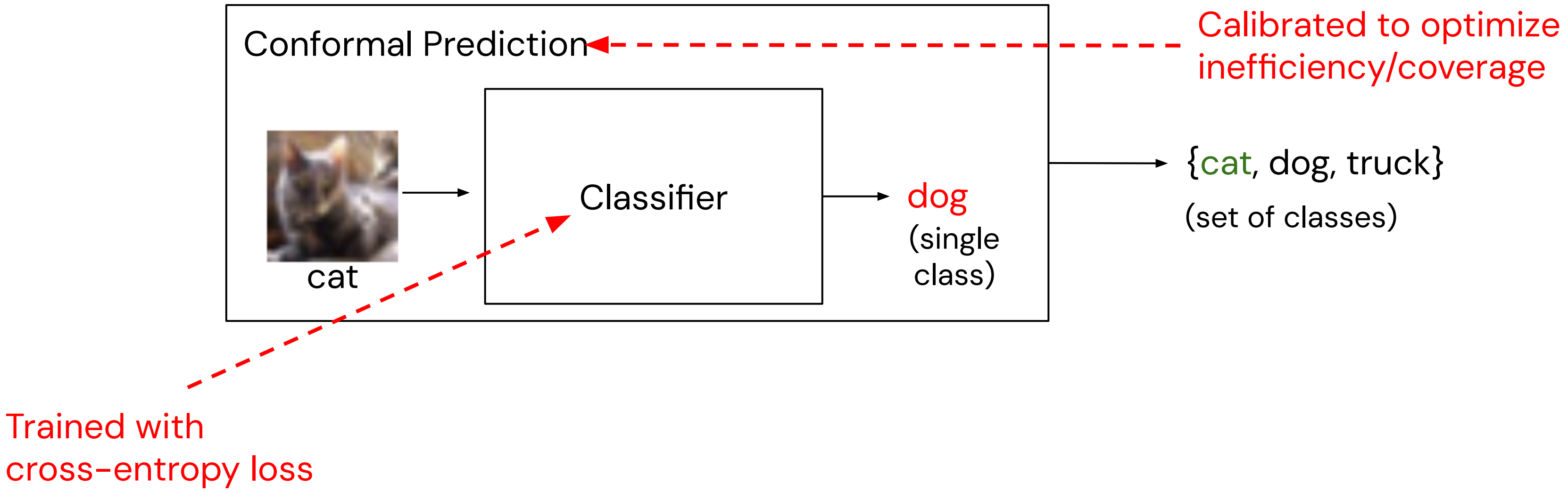


➔ True class is in the predicted confidence set with user-specified probability!

➔ Number of predicted classes = inefficiency

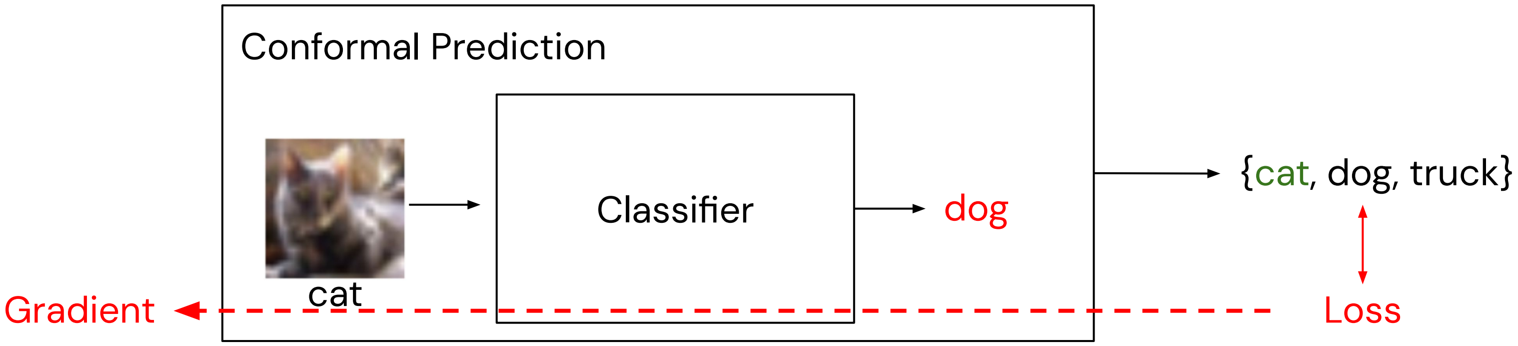
Overview and Motivation: Conformal Prediction

Training and conformalization objectives not aligned:



Overview and Motivation: Conformal *Training*

Conformal training = take conformal predictor into account during training:



→ Optimize arbitrary objectives defined on confidence sets

→ Obtain guaranteed coverage using any conformal predictor after training

DeepMind

Learning Optimal Conformal Classifiers

- ❑ Conformal Prediction
- ❑ Conformal Training
- ❑ Experimental Results
- ❑ Conclusion

Paper:
arxiv.org/abs/2110.09192

Conformal Prediction

For model $\pi_{\theta,y} \approx p(y|x)$, construct confidence sets $C_\theta(x) \subseteq [K] = \{1, \ldots, K\}$ such that:

$$P(y \in C_\theta(x)) \geq 1 - \alpha$$

- confidence level α user-specified

Conformal Prediction

For model $\pi_{\theta,y} \approx p(y|x)$, construct confidence sets $C_\theta(x) \subseteq [K] = \{1, \ldots, K\}$ such that:

$$P(y \in C_\theta(x)) \geq 1 - \alpha$$

- confidence level α user-specified
- *inefficiency* = average confidence set size $|C_\theta(x)|$ minimized

	{airplane}	{dog, cat}	{frog,horse,dog}	{cat,frog}	true class
coverage/inefficiency	yes/1	yes/2	no/3	yes/2	

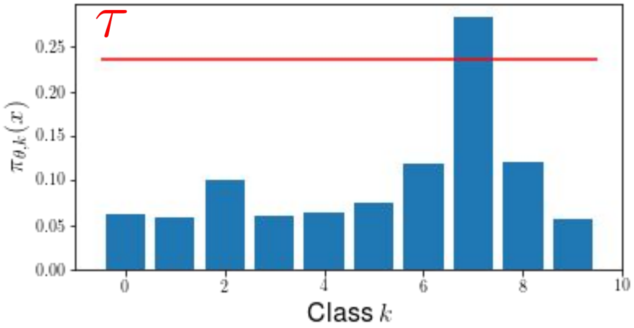
Example: Threshold Conformal Predictor

Split conformal prediction with two steps: *prediction* and *calibration*:

1. Prediction: define how confidence sets $C_\theta(x)$ are constructed,

$$C_\theta(x) := \{k \in [K] : E(x,k) := \pi_{\theta,k}(x) \geq \tau\}$$

with $E(x,k) := \pi_{\theta,k}(x)$ called conformity scores.



Mauricio Sadinle, Jing Lei, and Larry Wasserman. Least ambiguous set-valued classifiers with bounded error levels. Journal of the American Statistical Association (JASA), 114(525):223–234, 2019.

Example: Threshold Conformal Predictor

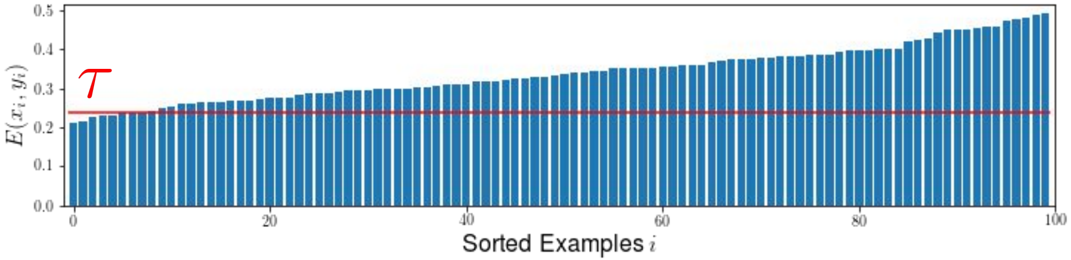
Split conformal prediction with two steps: *prediction* and *calibration*:

1. Prediction: define how confidence sets $C_\theta(x)$ are constructed.

$$C_\theta(x) := \{k \in [K] : E(x,k) := \pi_{\theta,k}(x) \geq \tau\}$$

2. Calibration: define threshold τ on held-out calibration set I_{cal}.

$$\tau = \alpha\text{-quantile of } \{E(x_i, y_i)\}_{i \in I_{\text{cal}}}$$



Mauricio Sadinle, Jing Lei, and Larry Wasserman. Least ambiguous set-valued classifiers with bounded error levels. Journal of the American Statistical Association (JASA), 114(525):223–234, 2019.

Example Results

Public

Inefficiency ↓ for different methods:

Dataset, α	Thr	APS	RAPS
CIFAR10, 0.05	**1.64**	2.06	1.74
CIFAR10, 0.01	**2.93**	3.30	3.06

82% accuracy on CIFAR10

Yaniv Romano, Matteo Sesia, and Emmanuel J. Candes. Classification with valid and adaptive coverage. In Advances in Neural Information Processing Systems (NIPS), 2020.
Anastasios Nikolas Angelopoulos, Stephen Bates, Michael I. Jordan, Jitendra Malik: Uncertainty Sets for Image Classifiers using Conformal Prediction. ICLR 2021

Example Results

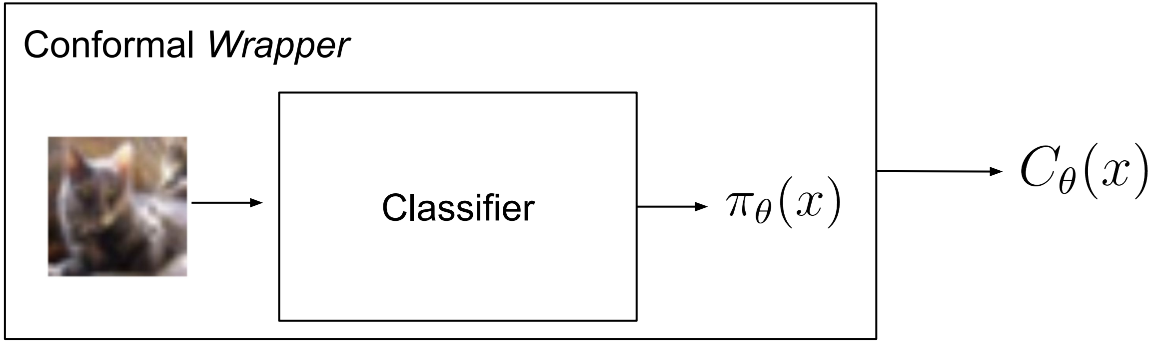
Inefficiency ↓ for different methods:

Dataset,	Thr	APS	RAPS
CIFAR10, 0.05	**1.64**	2.06	1.74
CIFAR10, 0.01	**2.93**	3.30	3.06
CIFAR100, 0.01	**10.63**	16.62	14.42

82% accuracy on CIFAR10

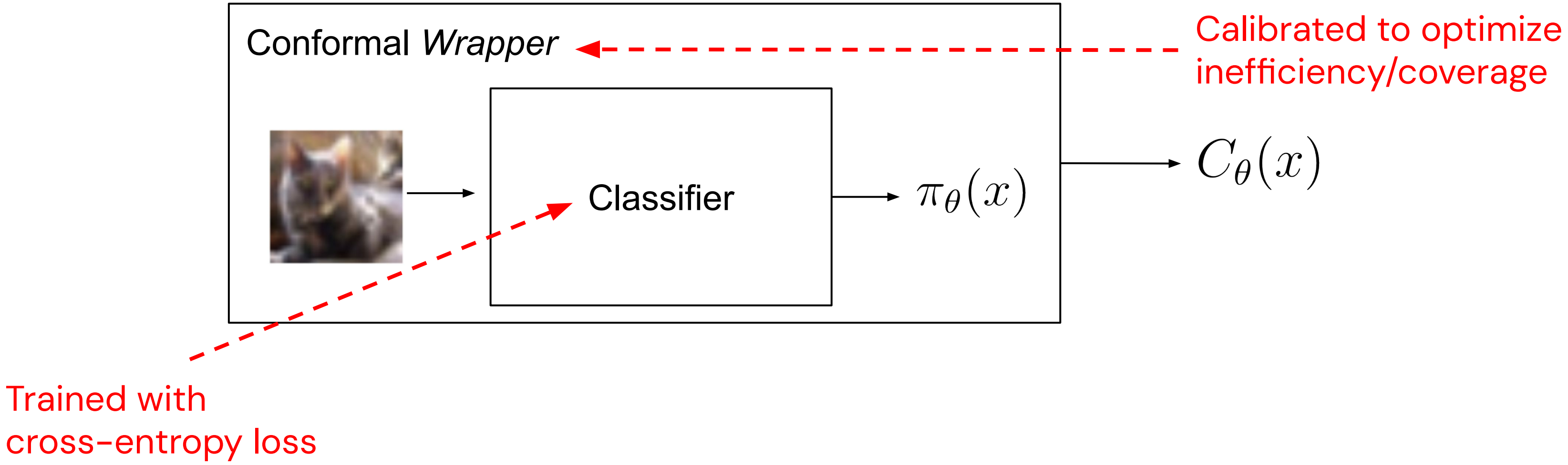
Training of Classifier *with* Conformal Wrapper

Conformal prediction is typically applied *after* training:



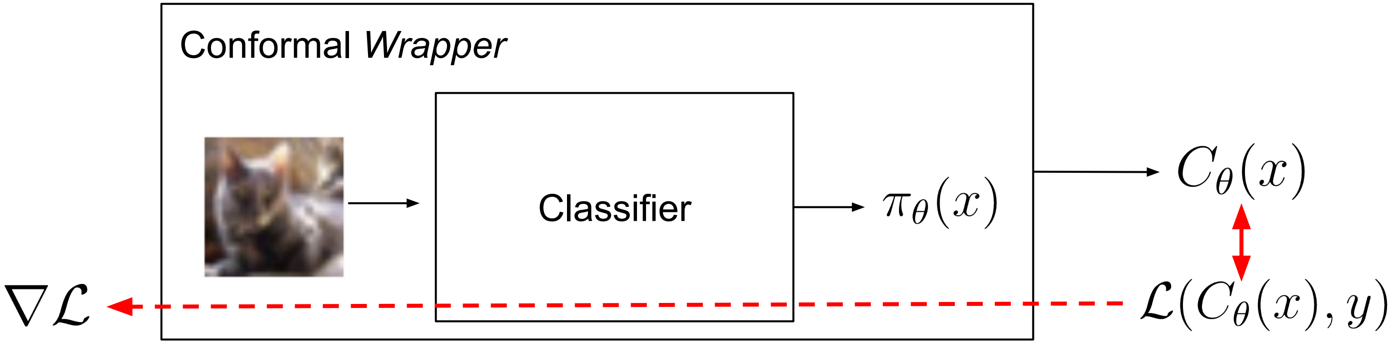
Training of Classifier *with* Conformal Wrapper

Conformal prediction is typically applied *after* training:



Training of Classifier *with* Conformal Wrapper

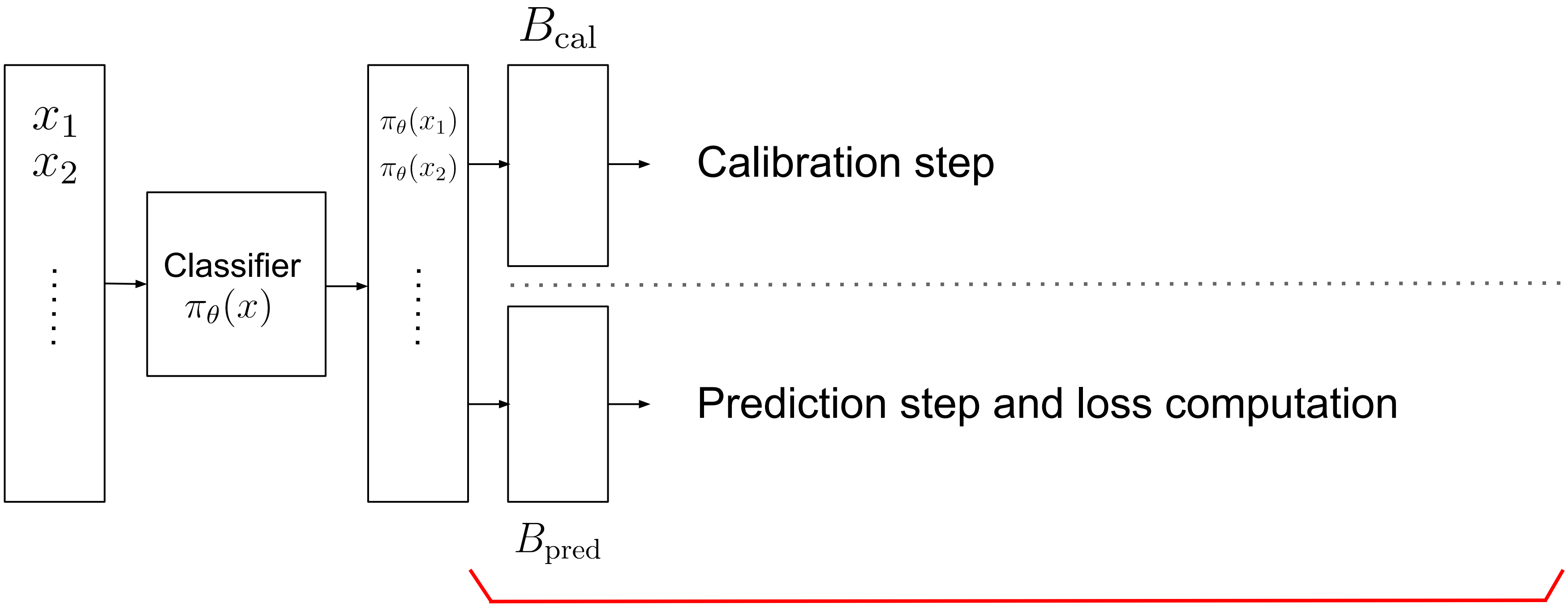
Conformal prediction is typically applied *after* training:



- ➔ Preserve coverage guarantee
- ➔ Independent of conformal predictor used at test time

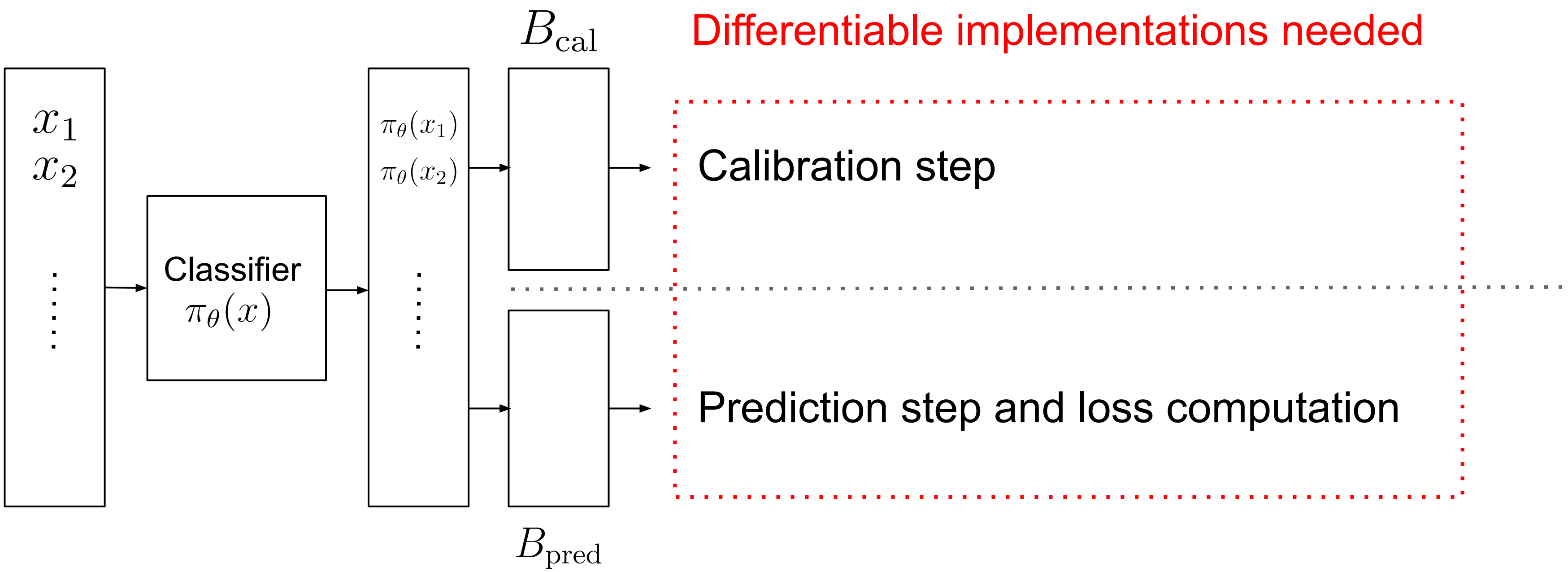
Conformal Training

Conformal Training



"Simulate" conformal prediction on each mini-batch

Conformal Training



Differentiable Conformal Prediction

Make both prediction and calibration steps differentiable:

1. Thresholding implemented using sigmoid function σ and temperature T

$$C_{\theta,k}(x) := \sigma((E(x,k) - \tau)/T) \in [0,1]$$

Differentiable Conformal Prediction

Make both prediction and calibration steps differentiable:

1. Thresholding implemented using sigmoid function σ and temperature T

$$C_{\theta,k}(x) := \sigma((\pi_{\theta,k}(x) - \tau)/T) \in [0, 1]$$

differentiable conformity score

Differentiable Conformal Prediction

Make both prediction and calibration steps differentiable:

1. Thresholding implemented using sigmoid function σ and temperature T

$$C_{\theta,k}(x) := \sigma((\pi_{\theta,k}(x) - \tau)/T) \in [0, 1]$$

interpreted as "soft" assignments

Differentiable Conformal Prediction

Public

Make both prediction and calibration steps differentiable:

1. Thresholding implemented using sigmoid function σ and temperature T

$$C_{\theta,k}(x) := \sigma((\pi_{\theta,k}(x) - \tau)/T) \in [0, 1]$$

2. Calibration using a smooth-sorter to compute the α-quantile.

Differentiable Conformal Prediction

Public

Make both prediction and calibration steps differentiable:

1. Thresholding implemented using sigmoid function σ and temperature T

$$C_{\theta,k}(x) := \sigma((\pi_{\theta,k}(x) - \tau)/T) \in [0, 1]$$

2. Calibration using a smooth-sorter to compute the α -quantile.

```
def smooth_predict_threshold(
    probabilities: jnp.ndarray, tau: float, temperature: float) -> jnp.ndarray:
  """Smooth implementation of prediction step for Thr."""
  return jax.nn.sigmoid((probabilities - tau) / temperature)

def smooth_calibrate_threshold(
    probabilities: jnp.ndarray, labels: jnp.ndarray,
    alpha: float, dispersion: float) -> float:
  """Smooth implementation of the calibration step for Thr."""
  conformity_scores = probabilities[jnp.arange(probabilities.shape[0]), labels.astype(int)]
  return smooth_quantile(array, dispersion, (1 + 1./array.shape[0]) * alpha)
```

Differentiable Conformal Prediction

Public

Make both prediction and calibration steps differentiable:

1. Thresholding implemented using sigmoid function σ and temperature T

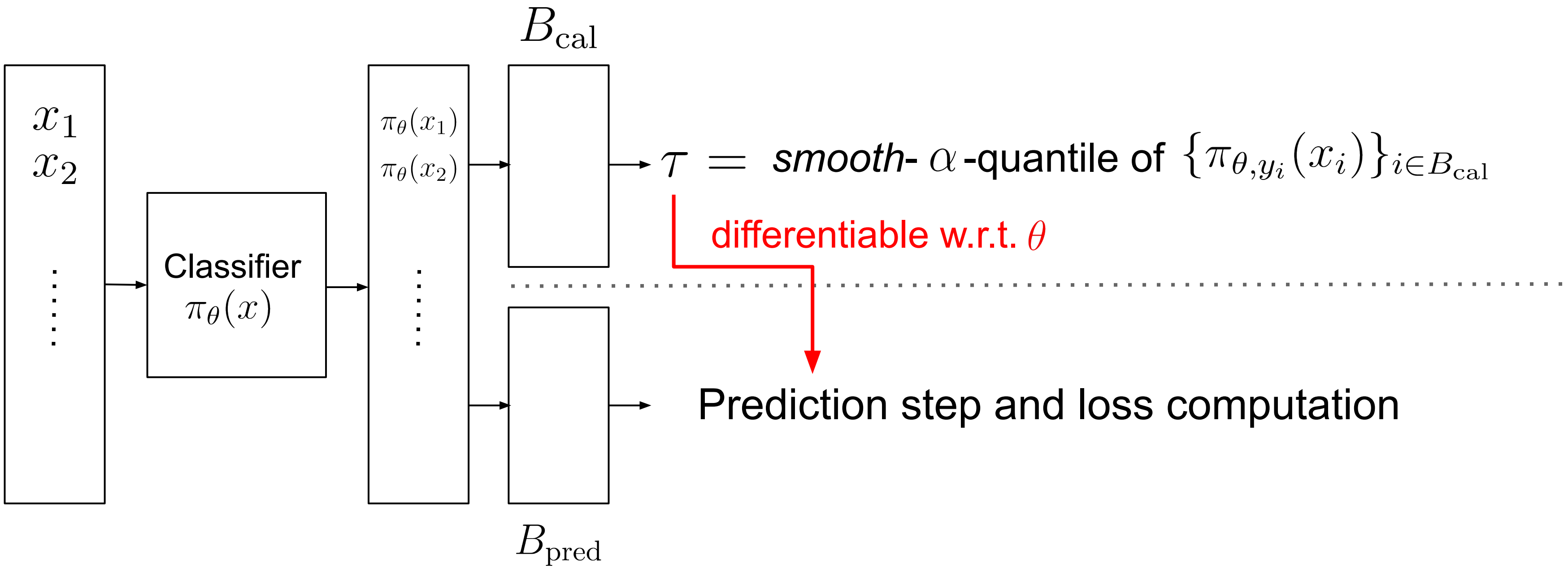
$$C_{\theta,k}(x) := \sigma((\pi_{\theta,k}(x) - \tau)/T) \in [0, 1]$$

2. Calibration using a smooth-sorter to compute the α-quantile.

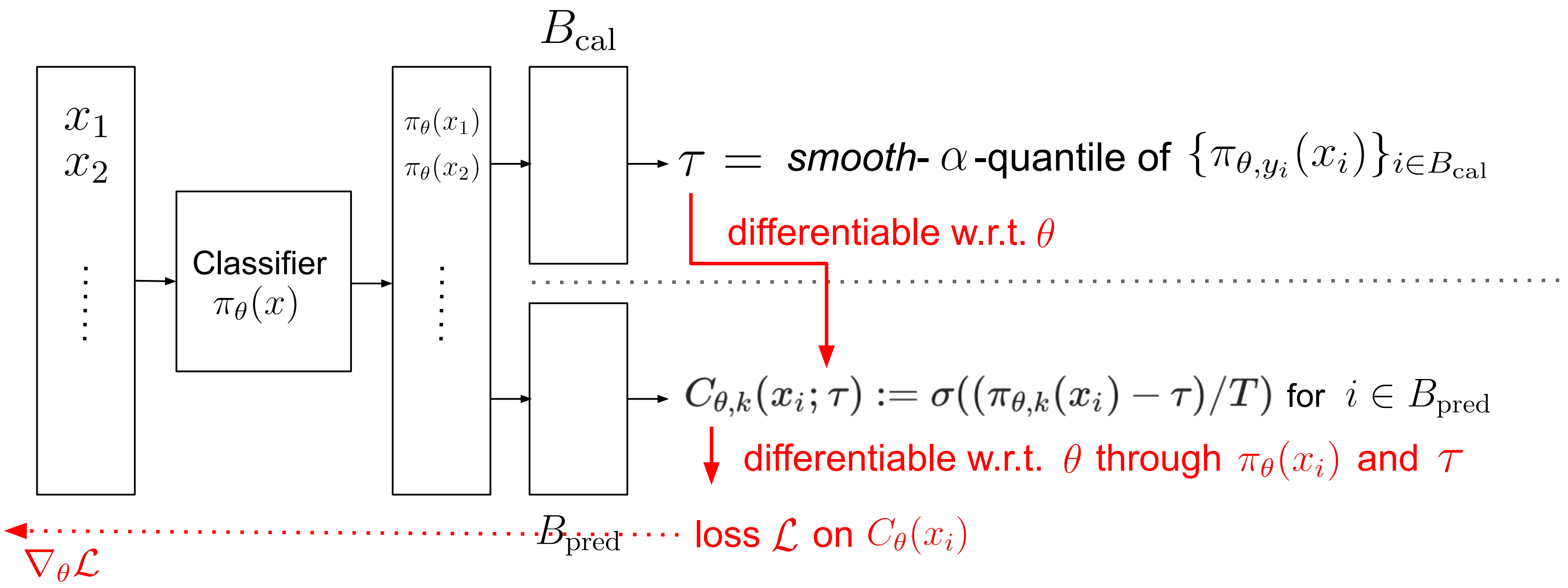
```
def smooth_predict_threshold(
    probabilities: jnp.ndarray, tau: float, temperature: float) -> jnp.ndarray:
  """Smooth implementation of prediction step for Thr."""
  return jax.nn.sigmoid((probabilities - tau) / temperature)

def smooth_calibrate_threshold(
    probabilities: jnp.ndarray, labels: jnp.ndarray,
    alpha: float, dispersion: float) -> float:
  """Smooth implementation of the calibration step for Thr."""
  conformity_scores = probabilities[jnp.arange(probabilities.shape[0]), labels.astype(int)]
  return smooth_quantile(array, dispersion, (1 + 1./array.shape[0]) * alpha)
```

→ Other differentiable conformity scores possible – e.g., APS.

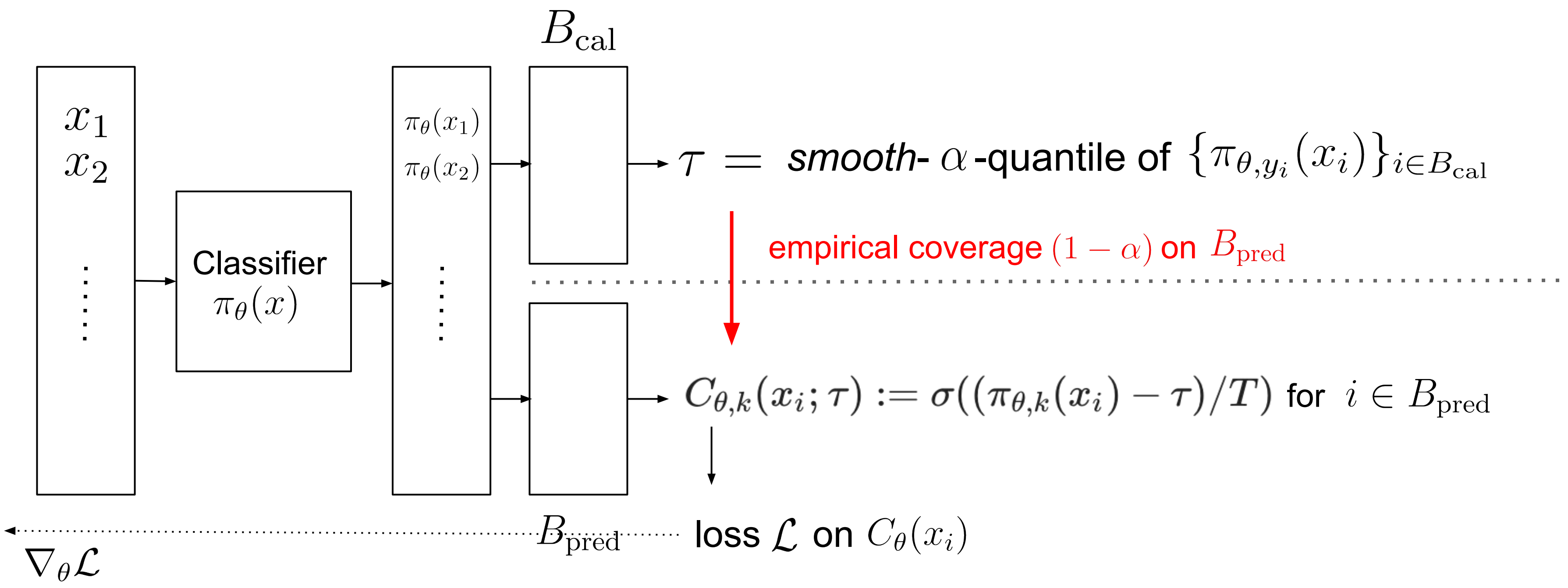
Conformal Training



Conformal Training



Conformal Training



➔ Re-calibrate at test time to obtain coverage guarantee!

Objectives

Public

Ⓐ Reducing inefficiency:

- Reduce overall uncertainty
- Reduce *class-conditional* uncertainty

Ⓑ Influencing the composition of confidence sets:

- Avoiding *coverage confusion*
- Reducing *mis-coverage*

Why Reduce Inefficiency?

Remember:

- Coverage is guaranteed
- Inefficiency reflects uncertainty

{frog,dog,deer,horse} {frog,dog,deer} {frog,deer} {deer}

reduced inefficiency = lower uncertainty translates to better resource/time usage to users

Optimizing Inefficiency

Train to directly reduce inefficiency:

$$\Omega(C_\theta(x)) = \sum_{k=1}^{K} C_{\theta,k}(x)$$

- $C_{\theta,k}(x) \in [0, 1]$ interpreted as “soft assignments”
- can be seen as smooth approximation of $\mathbb{E}[|C_\theta(x)|]$
- no loss on true label y as empirical coverage close to $(1 - \alpha)$

Reducing Inefficiency: Results

Public

Inefficiency ↓ for $\alpha = 0.01$:			
CP at test time:	Thr		
Dataset	Cross-entropy baseline	ConfTr (ours)	
MNIST	2.23	**2.11** (-5.4%)	
F-MNIST	2.05	**1.67** (-18.5%)	
EMNIST (K = 52)	2.66	**2.49** (-6.4%)	
CIFAR10	2.93	**2.84** (-3.1%)	
CIFAR100	10.63	**10.44** (-1.8%)	

Reducing Inefficiency: Results

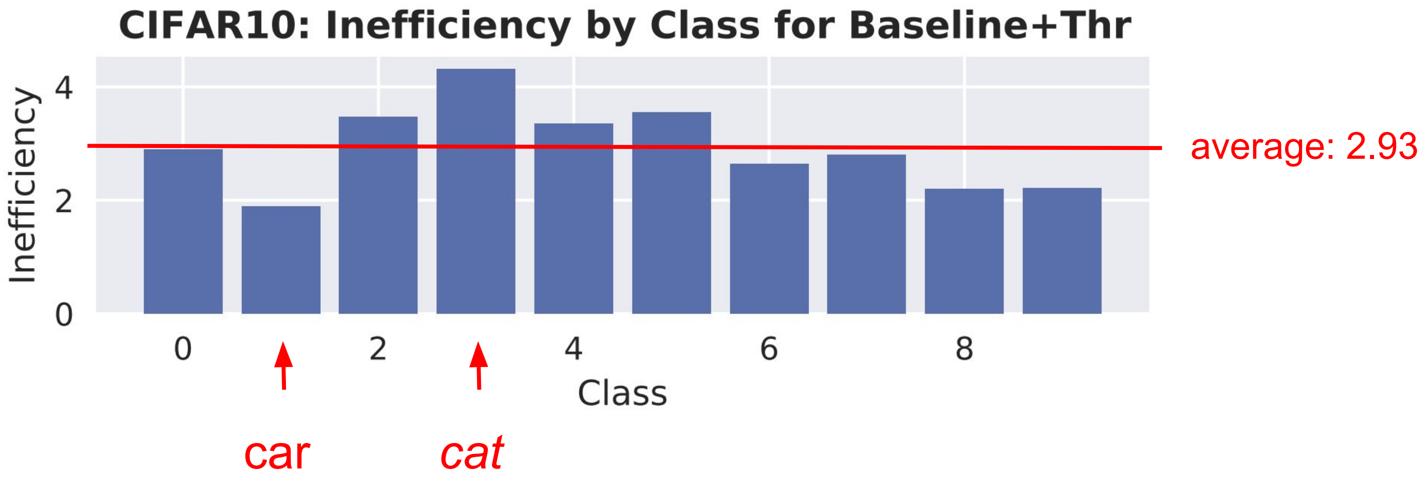
Public

Inefficiency ↓ for $\alpha = 0.01$:				
CP at test time:	Thr		APS	
Dataset	Cross-entropy baseline	ConfTr (ours)	Cross-entropy baseline	ConfTr (ours)
MNIST	2.23	**2.11** (-5.4%)	2.50	**2.14** (-14.14%)
F-MNIST	2.05	**1.67** (-18.5%)	2.36	**1.72** (-27.1%)
EMNIST (K = 52)	2.66	**2.49** (-6.4%)	4.23	**2.87** (-32.2%)
CIFAR10	2.93	**2.84** (-3.1%)	3.30	**2.93** (-11.1%)
CIFAR100	10.63	**10.44** (-1.8%)	16.62	**12.73** (-23.4%)

Inefficiency Distribution

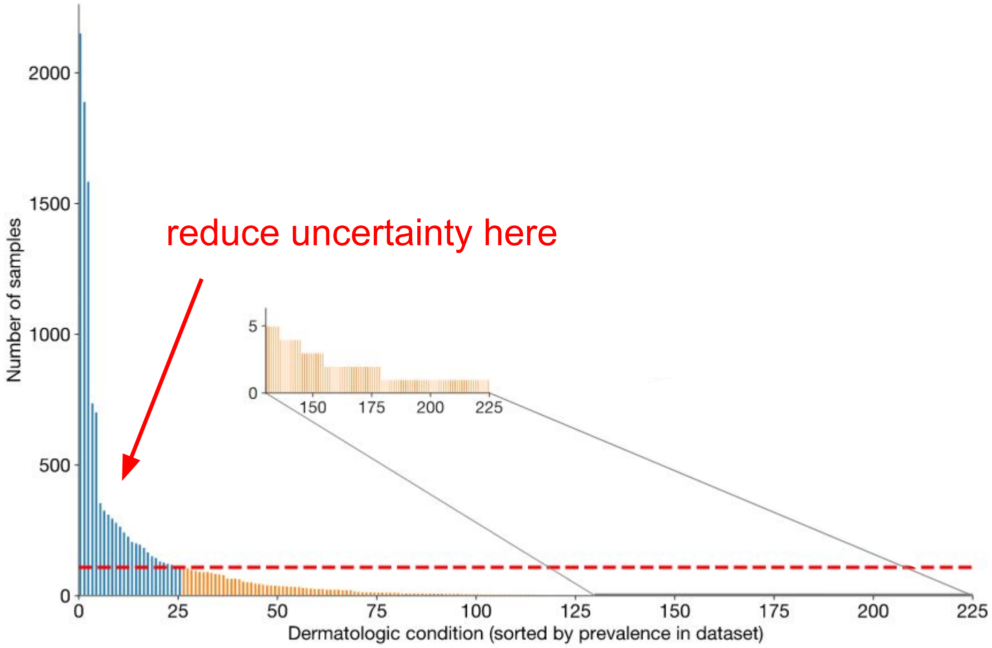
Public

Inefficiency ↓ distributed very differently across classes:



Reducing Class-Conditional Inefficiency

- Reduce inefficiency for “easy” / low-risk classes



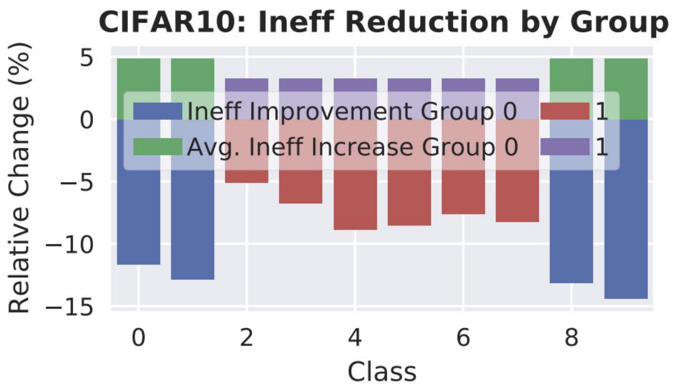
Roy et al. Does your dermatology classifier know what it doesn't know? Detecting the long-tail of unseen conditions. Medical Image Anal., 2022.

Results: CIFAR10

- Possible inefficiency improvement per class (in %)
- Cost in terms of average inefficiency increase across classes (in %)

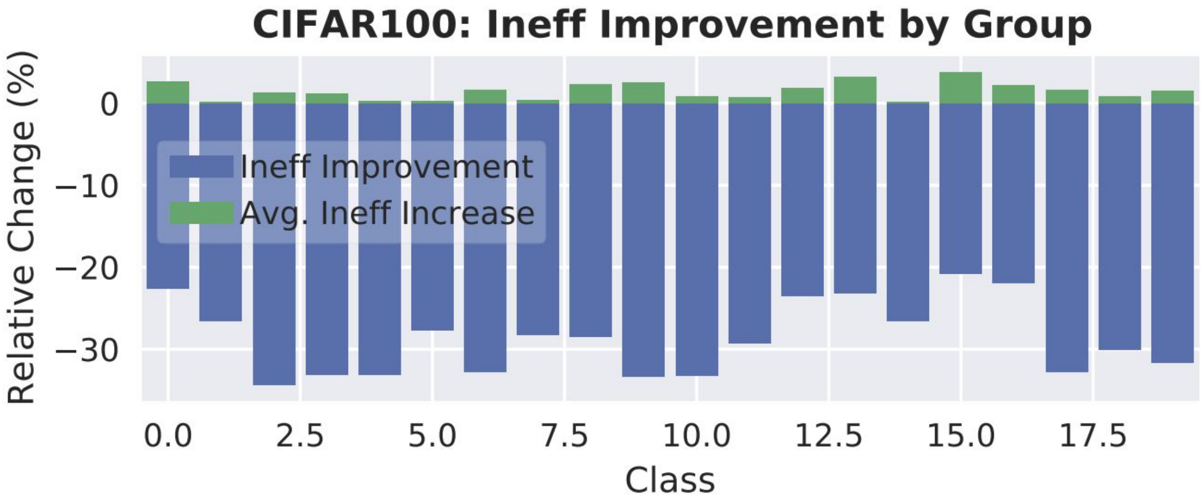
Results: CIFAR10

- Possible inefficiency improvement per class (in %)
- Cost in terms of average inefficiency increase across classes (in %)



More on Class-Conditional Inefficiency

- Possible inefficiency improvement per class (in %)
- Cost in terms of average inefficiency increase across classes (in %)



Objectives

Ⓐ Reducing inefficiency:

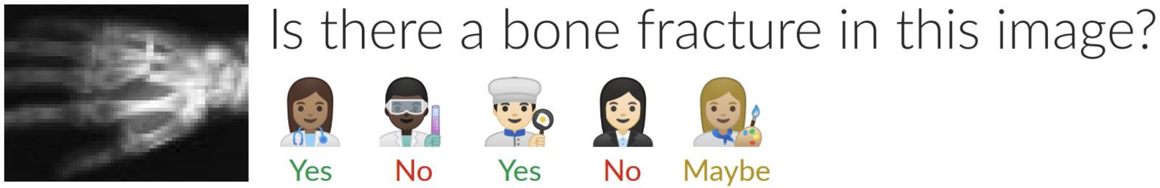
- Reduce overall uncertainty
- Reduce *class-conditional* uncertainty

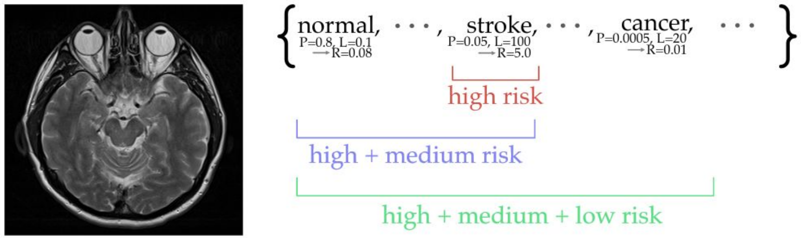
Ⓑ Influencing the composition of confidence sets:

- Avoiding *coverage confusion*
- Reducing *mis-coverage*

Beyond Reducing Inefficiency

- Shape composition of confidence sets:
  - Avoid confusion of specific, easily confused classes
  - Avoid mixing classes of different categories





Platanios et al. Learning from Imperfect Annotations. ArXiv, 2020.

Shaping Confidence Sets

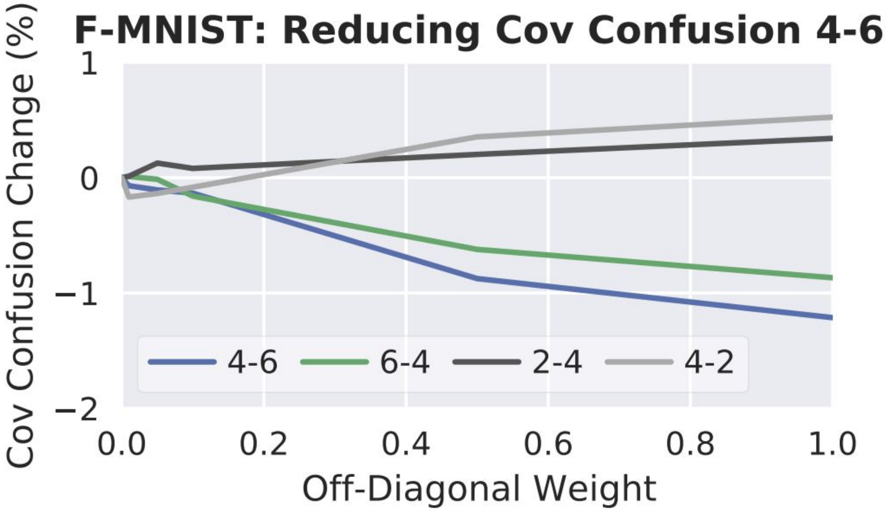
Which classes are actually included in $C_\theta(x)$?

$$\underbrace{\Omega(C_\theta(x))}_{\text{Ineff loss}} + \sum_{k=1}^{K} L_{y,k} \underbrace{[(1 - C_{\theta,k}(x))\delta[y = k]}_{\text{True class included}} + \underbrace{C_{\theta,k}(x)\delta[y \neq k]]}_{\text{Other classes \textit{not} included}}$$

- “just” enforces coverage with $L = I_K$
- use $L_{y,k} > 0$ to penalize class k occurring in confidence sets of class y

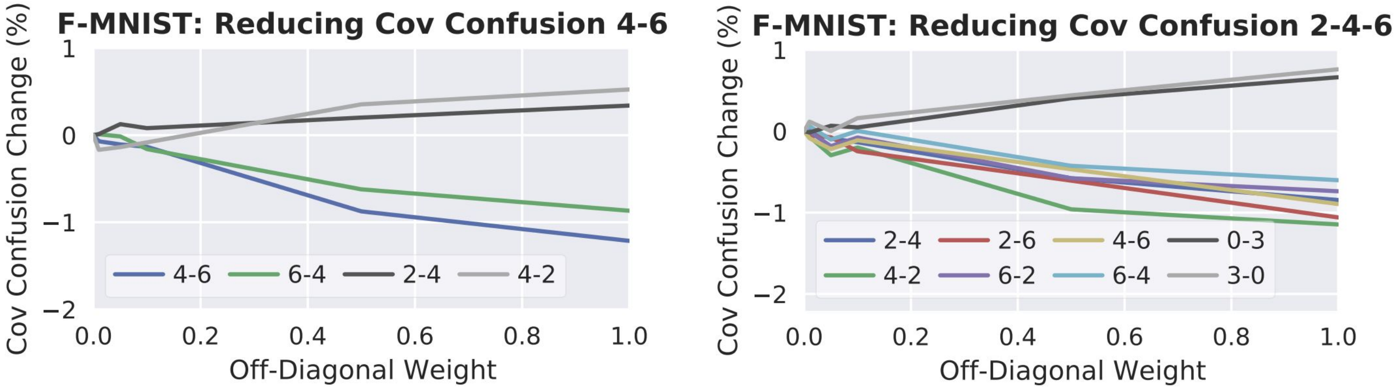
Example: Reduce Coverage Confusion

Reduce confusion between 4 (coat) and 6 (shirt) in confidence sets:



Example: Reduce Coverage Confusion

Reduce confusion between 2 (pullover), 4 and 6 in confidence sets:



Example: Reduce Mis-Coverage

Avoid natural and human-made classes in the same confidence sets:

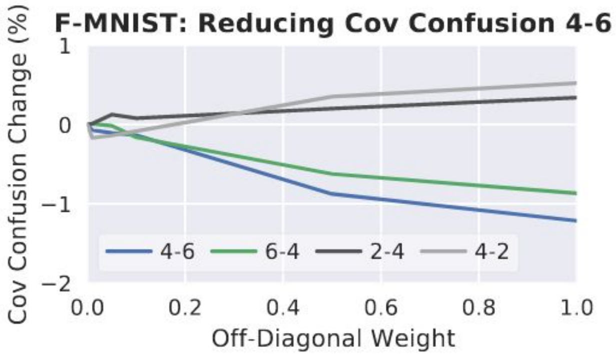
CIFAR100	Inefficiency	% *natural* classes in human-made confidence sets	% human-made classes in *natural* confidence sets
ConfTr	**10.44**	40.09	29.60
$L_{\text{human-made,natural}} > 0$	16.50	**15.77**	70.26
$L_{\text{natural,human-made}} > 0$	11.35	45.37	**17.56**

Conclusion: Conformal Training

= end-to-end training of classifier and conformal wrapper.

- retains coverage guarantee
- reduces inefficiency
- allows arbitrary, application-specific losses

Paper: arxiv.org/abs/2110.09192



Appendix

Example: Threshold Conformal Predictor

Public

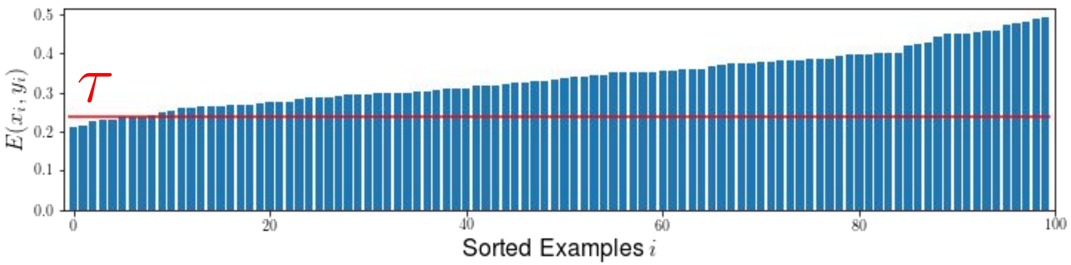
Split conformal prediction with two steps: *prediction* and *calibration*:

1. Prediction: define how confidence sets $C_\theta(x)$ are constructed.

$$C_\theta(x) := \{k \in [K] : E(x,k) := \pi_{\theta,k}(x) \geq \tau\}$$

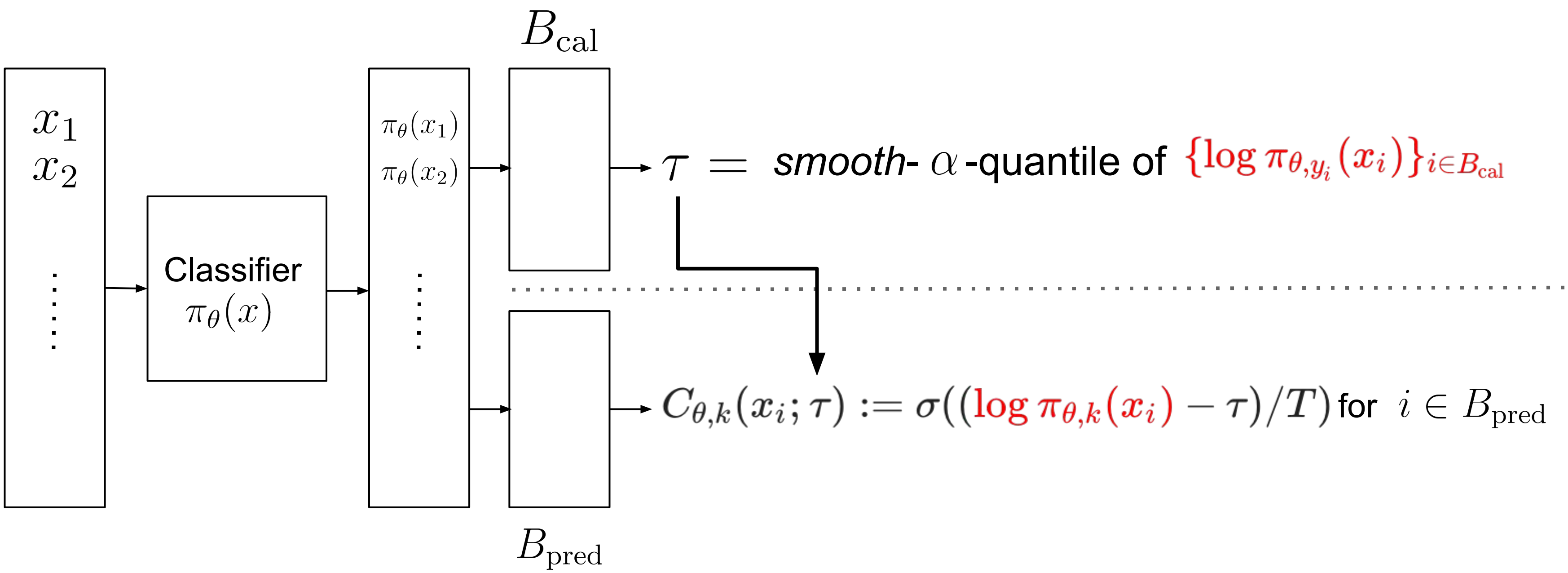
2. Calibration: define threshold τ on held-out calibration set I_{cal}.

$$\tau = \alpha \cdot (1 + 1/|I_{\text{cal}}|) \text{-quantile of } \{E(x_i, y_i)\}_{i \in I_{\text{cal}}}$$



Mauricio Sadinle, Jing Lei, and Larry Wasserman. Least ambiguous set-valued classifiers with bounded error levels. Journal of the American Statistical Association (JASA), 114(525):223–234, 2019.

Conformal Training



Smooth Conformal Prediction

Prediction step:

function PREDICT($\pi_\theta(x), \tau$)
  compute $E_\theta(x,k), k \in [K]$
  return $C_\theta(x;\tau) = \{k : E_\theta(x,k) \geq \tau\}$

Calibration step:

function CALIBRATE($\{(\pi_\theta(x_i), y_i\}_{i=1}^{n}, \alpha$)
  compute $E_\theta(x_i, y_i), i=1,\ldots,n$
  return QUANTILE($\{E_\theta(x_i, y_i)\}, \alpha(1 + 1/n)$)

Smooth implementation:

function SMOOTHPRED($\pi_\theta(x), \tau, T=1$)
  return $C_{\theta,k}(x;\tau) = \sigma(\frac{(E_\theta(x,k)-\tau)}{T}), k \in [K]$
function SMOOTHCAL($\{(\pi_\theta(x_i), y_i\}_{i=1}^{n}, \alpha$)
  return SMOOTHQUANT($\{E_\theta(x_i, y_i)\}, \alpha(1+\frac{1}{n})$)

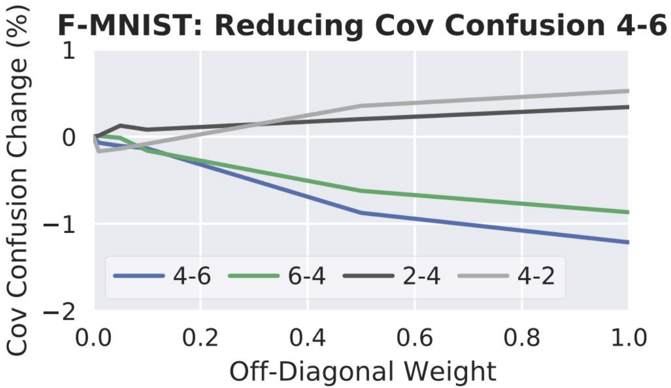
Conformal Training: Algorithm

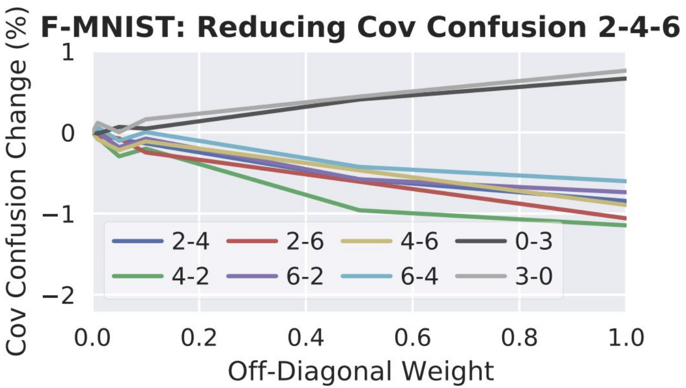
function $\text{ConformalTraining}(\alpha, \lambda{=}1)$
  for mini-batch B **do**
    randomly split batch $B_{\text{cal}} \uplus B_{\text{pred}} = B$
    {"On-the-fly" calibration on B_{cal}:}
    $\tau = \text{SmoothCal}(\{(\pi_\theta(x_i), y_i)\}_{i \in B_{\text{cal}}}, \alpha)$
    {Prediction only on $i \in B_{\text{pred}}$:}
    $C_\theta(x_i;\tau) = \text{SmoothPred}(\pi_\theta(x_i), \tau)$
    {*Optional* classification loss:}
    $\mathcal{L}_B = 0$ or $\sum_{i \in B_{\text{pred}}} \mathcal{L}(C_\theta(x_i;\tau), y_i)$
    $\Omega_B = \sum_{i \in B_{\text{pred}}} \Omega(C_\theta(x_i;\tau))$
    $\Delta = \nabla_\theta \, ^1/_{|B_{\text{pred}}|}(\mathcal{L}_B + \lambda \Omega_B)$
    update parameters θ using Δ

Coverage Confusion

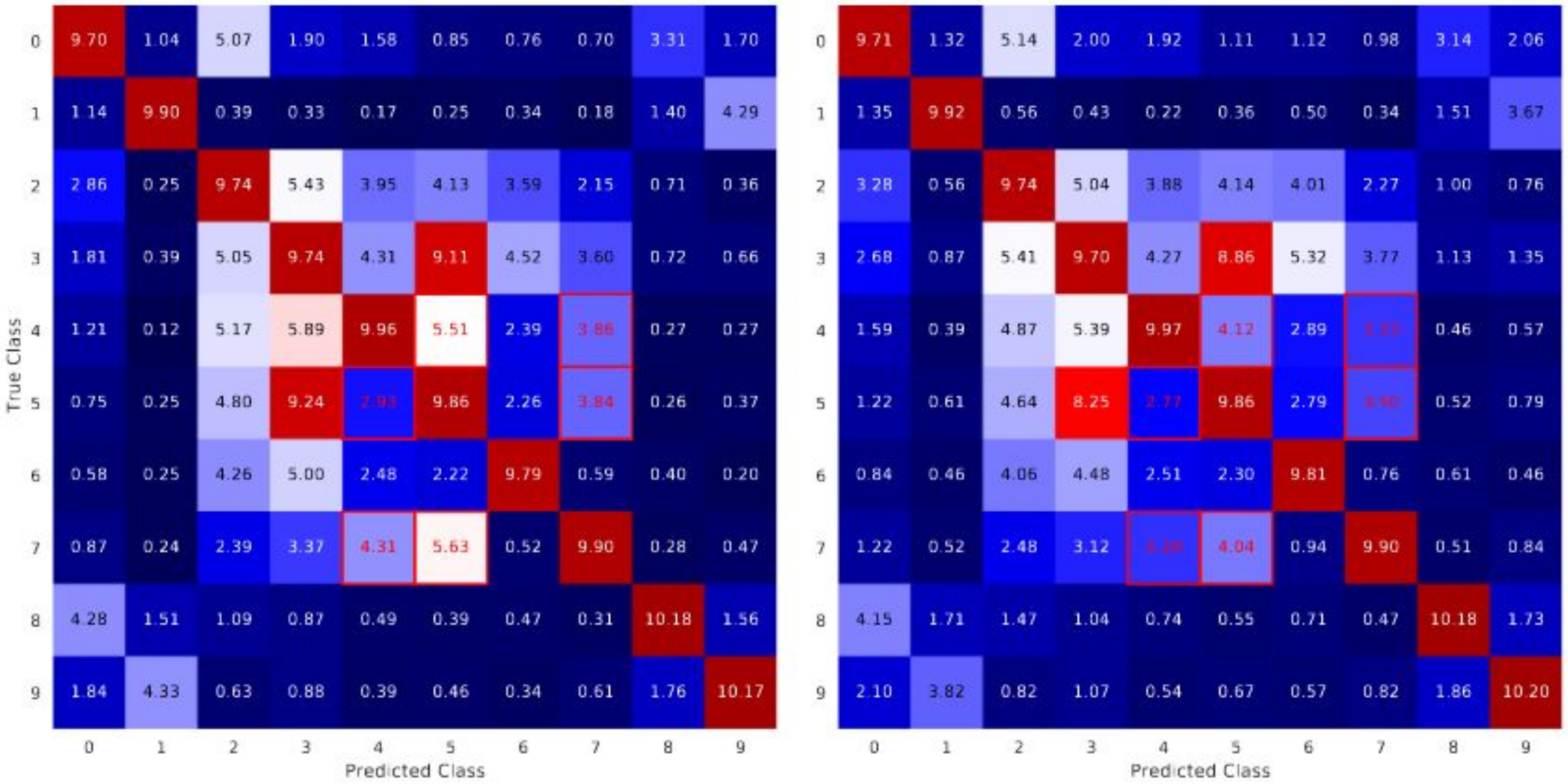
Formal definition:

$$\Sigma_{y,k} := \frac{1}{I_{\text{test}}} \sum_{i \in I_{\text{test}}} \delta[y_i = y \wedge k \in C(x_i)]$$





Coverage Confusion: Example



Mis-Coverage

Public

Based on two disjoint subsets of classes $K_0 \cap K_1 = \emptyset$:

$$\text{MisCover}_{0\to 1} = \frac{1}{\sum_{i \in I_{\text{test}}} \delta[y_i \in K_0]} \sum_{i \in I_{\text{test}}} \delta[y_i \in K_0 \wedge (\exists k \in K_1 : k \in C(x_i))]$$

CIFAR10: K_0= 3 (“cat”) vs. K_1= Others
CIFAR100: K_0= “human-made vs. K_1= “natural”

	CIFAR10			CIFAR100		
		MisCover ↓			MisCover ↓	
Method	Ineff	0→1	1→0	Ineff	0→1	1→0
ConfTr	**2.84**	98.92	36.52	**10.44**	40.09	29.6
L_{K_0,K_1}=1	2.89	**91.60**	34.74	16.50	**15.77**	70.26
L_{K_1,K_0}=1	2.92	97.36	**26.43**	11.35	45.37	**17.56**

Binary Datasets