A. Example Augmented Algorithms

We detail the augmented examples from the generalization section here. We provide the modified pseudocode based on published papers and publically available code for the evaluated algorithms Co-Teaching+ [34], M-DYR-H [1], and DivideMix [14]. We bold the region where augmentation is inserted. No hyperparamters were changed in any of the experiments. All models used are the same as those used in the originally published papers.

A.1. Augmented CoTeaching+

We provde the full implementation of the Co-Teaching+ [34] algorithm with our augmentations below. Co-Teaching+ is similar to the original Co-Teaching [10], but adds a different disagreement component when the two network predictions are not similar. If the predictions are similar, training is conducted on the lower loss samples. We find that adding strong augmentation to this part of the training improves performance. Co-teaching uses a threshold $R(e)$ instead of a mixture model fitting to take advantage of the memorization effect. The two network setup that it uses is an effective technique in existing algorithms.

Algorithm 2: Augmented Co-Teaching+

```
Input θ^(1) and θ^(2), training dataset (X, Y), learning rate
η, fixed τ, epoch T_k and T_max, strong augmentation
function Augment.
$\theta = \text{WarmUp}(\mathcal{X}, \mathcal{Y}, \theta)$
while $e < T_{\max}$ do
    for $b = 1$ to $B$ do
        Select prediction disagreement $\bar{x_b}'$ from batch $x_b$;
        if $|\bar{x_b}'| > 0$ then
            $\bar{x_b}'^{(1)} = \arg\min_{\bar{x_b}':|\bar{x_b}'| \geq \lambda(e)|\bar{x_b}'|} \ell(\bar{x_b}'; \theta^{(1)})$;
            $\bar{x_b}'^{(2)} = \arg\min_{\bar{x_b}':|\bar{x_b}'| \geq \lambda(e)|\bar{x_b}'|} \ell(\bar{x_b}'; \theta^{(2)})$;
            $\theta^{(1)} = \theta^{(1)} - \eta \nabla \ell(\bar{x_b}'^{(2)}; \theta^{(1)})$;
            $\theta^{(2)} = \theta^{(2)} - \eta \nabla \ell(\bar{x_b}'^{(1)}; \theta^{(2)})$;
        else
            $x_b^{(1)} = \arg\min_{x_b':|x_b'| \geq R(e)|x_b|} \ell(x_b; \theta^{(1)})$;
            $x_b^{(2)} = \arg\min_{x_b':|x_b'| \geq R(e)|x_b|} \ell(x_b; \theta^{(2)})$;
            $\theta^{(1)} = \theta^{(1)} - \eta \nabla \ell(\textbf{Augment}(x_b^{(1)}); \theta^{(1)})$;
            $\theta^{(2)} = \theta^{(2)} - \eta \nabla \ell(\textbf{Augment}(x_b^{(2)}); \theta^{(2)})$;
    Update $R(e) = 1 - \min\left\{\frac{e}{T_k}\tau, \tau\right\}$;
Output $\theta^{(1)}$ and $\theta^{(2)}$.
```

A.2. Augmenting M-DYR-H

We provide the full implementation of M-DYR-H algorithm from [1]. M-DYR-H warmup, and uses mixup training on input batches to obtain strong results. The loss is weighted using a BMM that is fit to the loss from previous epochs. During warmup, we leave the existing weak-augmentations in place. For the pseudolabel prediction z_b as well as the BMM modelling W, we use weak augmentations. We insert strong augmentations during the mixup process which is independent of what the network uses to model the losses. We find that this can improve performance.

Algorithm 3: Augmented M-DYR-H

```
Input: $\theta$, training dataset $(\mathcal{X}, \mathcal{Y})$, Beta distribution
  parameter $\alpha$ for mixup, strong augmentation function
  Augment.
$\theta = \text{WarmUp}(\mathcal{X}, \mathcal{Y}, \theta)$
while $e < \text{MaxEpoch}$ do
    $\mathcal{W} = \text{BMM}(\mathcal{X}, \mathcal{Y}, \theta)$
    for $b = 1$ to $B$ do
        $z_b = \text{p}_{\text{model}}(x_b; \theta)$
        $w_b$ = compute_batch_probs$(x_b, \mathcal{W}, y_b)$
        $x_b^m, y_b^1, y_b^2, z_b^1, z_b^2, w_b^1, w_b^2, \lambda$ =
          mixup$(\textbf{Augment}(x_b), y_b, z_b, w_b)$
        $z_m = \text{p}_{\text{model}}(x_b^m; \theta)$
        $l_1 = (1 - w_b^1) \sum \text{NLLoss}(log(z_m), y_1^m)/|x_b|$
        $l_2 = w_b^1 \sum \text{NLLoss}(log(z_m), z_1^m)/|x_b|$
        $l_3 = (1 - w_b^2) \sum \text{NLLoss}(log(z_m), y_2^m)/|x_b|$
        $l_4 = w_b^2 \sum \text{NLLoss}(log(z_m), z_2^m)/|x_b|$
        $\mathcal{L} = \lambda(l_1 + l_2) + (1 - \lambda)(l_3 + l_4) + \lambda_r \mathcal{L}_{reg}$
        $\theta = \text{SGD}(\mathcal{L}, \theta)$
Output $\theta$.
```

A.3. Augmenting DivideMix

A full version of the algorithm outlined in DivideMix is provided here (Algorithm 4). The technique is a combination of co-training, MixUp, loss modeling, and is trained in a semi-supervised learning manner. Notation and algorithm are as presented in the original paper [14]. We insert our changes in bold.

Algorithm 4: Augmented DivideMix.

```
Input: $\theta^{(1)}$ and $\theta^{(2)}$, training dataset $(\mathcal{X}, \mathcal{Y})$, clean probability threshold $\tau$, number of augmentations $M$, augmentation policies
  Augment$_1$ and Augment$_2$, sharpening temperature $T$, unsupervised loss weight $\lambda_u$, Beta distribution parameter $\alpha$ for
  MixMatch.
$\theta^{(1)}, \theta^{(2)} = \mathrm{WarmUp}(\mathcal{X}, \mathcal{Y}, \theta^{(1)}, \theta^{(2)})$    // standard training (with confidence penalty)
while $e <$ MaxEpoch do
    $\mathcal{W}^{(2)} = \mathrm{GMM}(\mathcal{X}, \mathcal{Y}, \theta^{(1)})$    // model per-sample loss with $\theta^{(1)}$ to obtain clean probability for $\theta^{(2)}$
    $\mathcal{W}^{(1)} = \mathrm{GMM}(\mathcal{X}, \mathcal{Y}, \theta^{(2)})$    // model per-sample loss with $\theta^{(2)}$ to obtain clean probability for $\theta^{(1)}$
    for $k = 1, 2$ do    // train the two networks one by one
        $\mathcal{X}_e^{(k)} = \{(x_i, y_i, w_i) | w_i \geq \tau, \forall (x_i, y_i, w_i) \in (\mathcal{X}, \mathcal{Y}, \mathcal{W}^{(k)})\}$    // labeled training set for $\theta^{(k)}$
        $\mathcal{U}_e^{(k)} = \{x_i | w_i < \tau, \forall (x_i, w_i) \in (\mathcal{X}, \mathcal{W}^{(k)})\}$    // unlabeled training set for $\theta^{(k)}$
        for iter = 1 to num_iters do
            From $\mathcal{X}_e^{(k)}$, draw a mini-batch $\{(x_b, y_b, w_b); b \in (1, ..., B)\}$
            From $\mathcal{U}_e^{(k)}$, draw a mini-batch $\{u_b; b \in (1, ..., B)\}$
            for $b = 1$ to $B$ do
                $x^{desc} =$ Augment$_2$ $(x_b)$
                $u^{desc} =$ Augment$_2$ $(x_b)$
                for $m = 1$ to $M$ do
                    $\hat{x}_{b,m} = \mathrm{Augment}_1(x_b)$
                    $\hat{u}_{b,m} = \mathrm{Augment}_1(u_b)$
                $p_b = \frac{1}{M} \sum_m \mathrm{p}_{\mathrm{model}}(\hat{x}_{b,m}; \theta^{(k)})$    // average the predictions across augmentations of $x_b$
                $\bar{y}_b = w_b y_b + (1 - w_b) p_b$
                    // refine ground-truth label guided by the clean probability produced by the other network
                $\hat{y}_b = \mathrm{Sharpen}(\bar{y}_b, T)$    // apply temperature sharpening to the refined label
                $\bar{q}_b = \frac{1}{2M} \sum_m \left( \mathrm{p}_{\mathrm{model}}(\hat{u}_{b,m}; \theta^{(1)}) + \mathrm{p}_{\mathrm{model}}(\hat{u}_{b,m}; \theta^{(2)}) \right)$
                    // co-guessing: average the predictions from both networks across augmentations of $u_b$
                $\hat{q}_b = Sharpen(\bar{q}_b, T)$
                    // apply temperature sharpening to the guessed label
            // train using a different augmentation
            $\hat{\mathcal{X}} = \{(x, y) | x \in x^{desc}, y \in \hat{y}\}$    // train with different augmentation
            $\hat{\mathcal{U}} = \{(u, q) | u \in u^{desc}, q \in \hat{q}\}$    // train with different augmentation
            $\mathcal{L}_{\mathcal{X}}, \mathcal{L}_{\mathcal{U}} = \mathrm{MixMatch}(\hat{\mathcal{X}}, \hat{\mathcal{U}})$    // apply MixMatch
            $\mathcal{L} = \mathcal{L}_{\mathcal{X}} + \lambda_u \mathcal{L}_{\mathcal{U}} + \lambda_r \mathcal{L}_{\mathrm{reg}}$    // total loss
            $\theta^{(k)} = \mathrm{SGD}(\mathcal{L}, \theta^{(k)})$    // update model parameters
```