# Highlights of the NeurIPS 2023 unlearning challenge

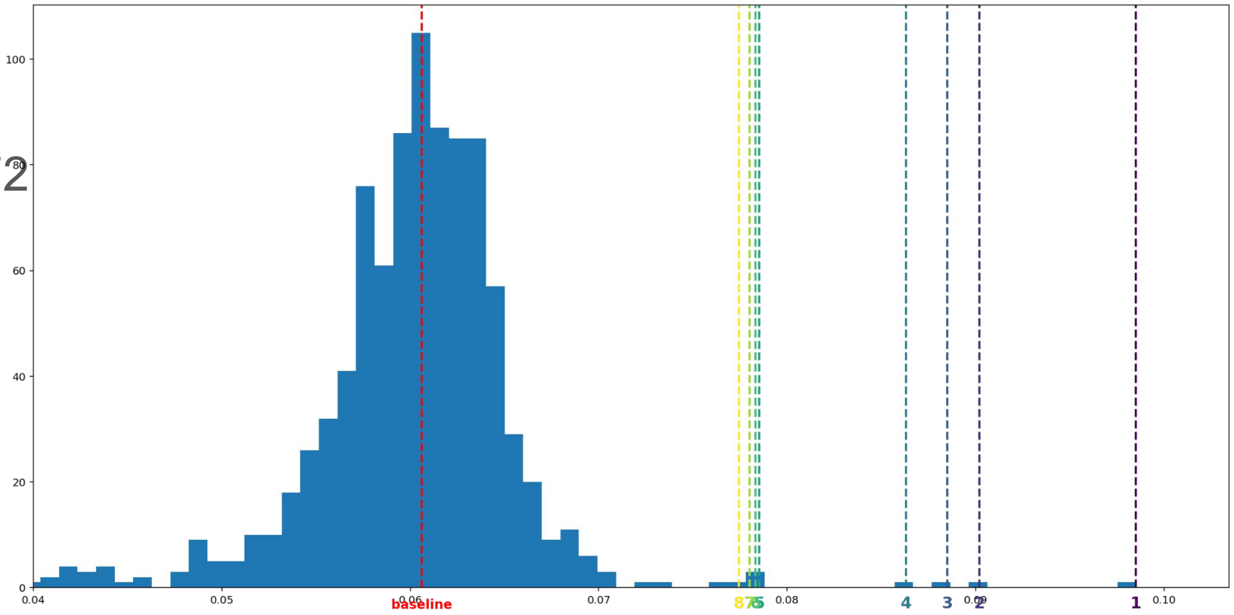
5,161 registrations

1,338 participants from 72 countries.
For 500 (including 44 in the top 100!), this was their first competition

1,121 teams

1,923 submissions

🤯



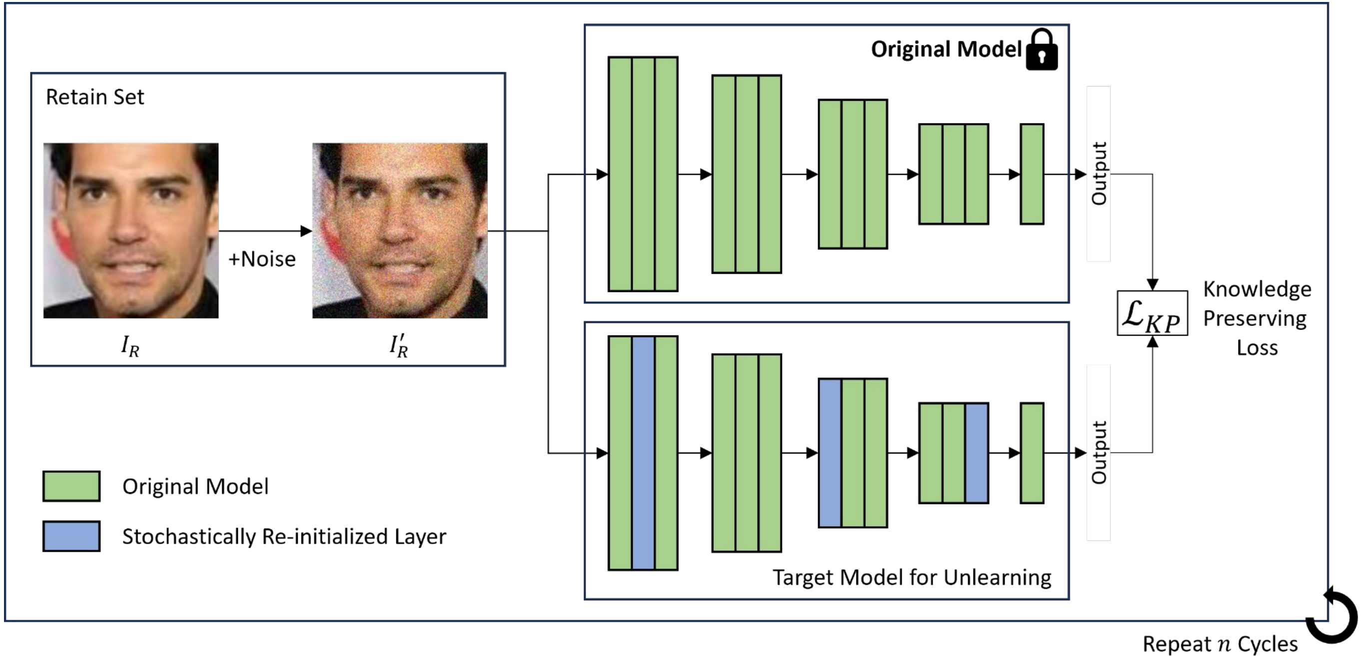

Leaderboard: 40% scored above baseline

# Top submissions

Prize Winners

| # | Team | Members |
|---|---|---|
| 1 | fanchuan | |
| 2 | [kookmin Univ] LD&BGW&KJH | |
| 3 | Seif Eddine Achour | |
| 4 | Sebastian Oleszko | |
| 5 | toshi_k & marvelworld | |
| 6 | Algorithmic Amnesiacs | |
| 7 | Jiaxi Sun | |
| 8 | Forget | |

# 8th place solution - Team Forget




(1) Forgetting phase: model parameters are stochastically selected and re-initialized.

- FC, Projection-shortcut layers are excluded from the selection pool.

(2) Remembering phase: knowledge preserving loss is calculated between the original model and the target unlearning model.

- Knowledge Preserving Loss:

$$\mathbb{E}\{|f_O(\mathbf{I}'_R) - f_U(\mathbf{I}'_R)|^2\}$$

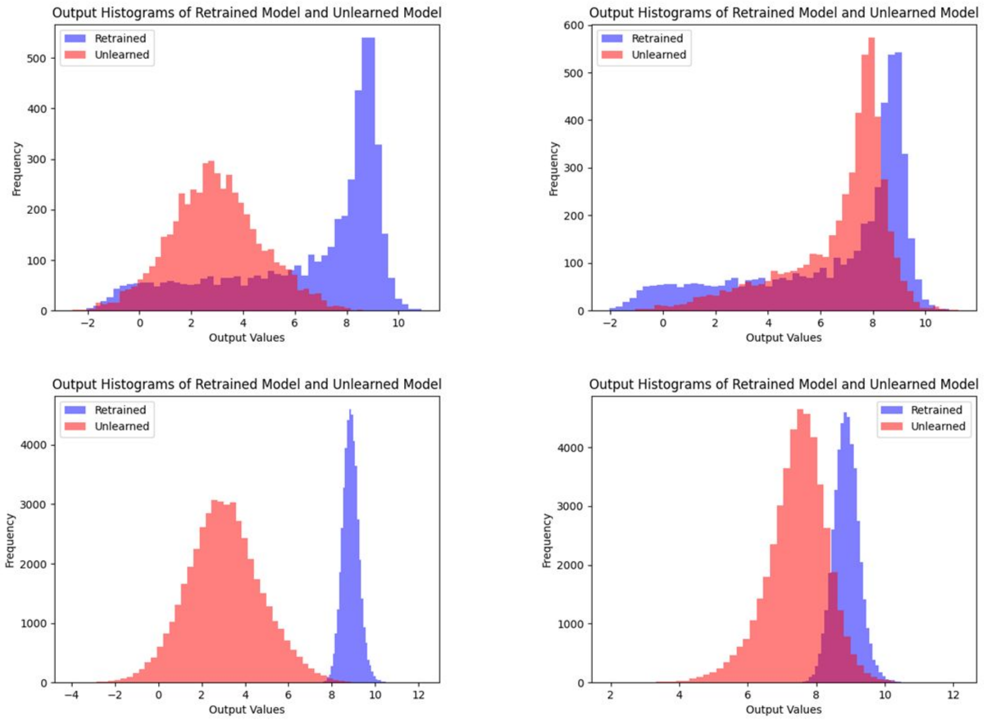
- Gaussian noise is added to the image as data augmentation.
- It reminds the target model about the retain set.

(3) Forgetting phase and Remembering phase are repeated for n cycles to enhance unlearning performance.

# 8th place solution - Team Forget

Forget Set

Retain Set




(a) CE (b) MSE

Figure. Comparison of logit distributions between CE loss and MSE loss

- Histograms of logits from retrained model and unlearned models are visualized.

- This observation is acquired from local experiments on CIFAR-10.

- MSE loss makes closer distributions than CE loss for both forget set and retain set.

# 8th place solution - Team Forget

- Gaussian Noise (σ=0.1) is the best data augmentation compared with other data augmentation techniques.

- Compared with CE loss and L1 loss, MSE loss demonstrates the best score.

- Additionally, increasing the cycles highly improves the performance.

- From these observations, we build the final submission.

Table 1. Comparison between different data augmentation techniques

| Input Data | Score↑ |
|---|---|
| Clean Image | 0.06172 |
| Vertical Flip | 0.02505 |
| Random Crop | 0.00001 |
| Cutout | 0.00001 |
| Image + Gaussian Noise ($\sigma = 0.1$) | **0.06532** |

Table 2. Comparison between different sigma of gaussian noise

| Input Data | Score↑ |
|---|---|
| Image + Gaussian Noise ($\sigma = 0.05$) | 0.06333 |
| Image + Gaussian Noise ($\sigma = 0.15$) | 0.05907 |
| Image + Gaussian Noise ($\sigma = 0.1$) | **0.06532** |

Table 3. Comparison between different loss functions

| Loss Function | Score↑ |
|---|---|
| CE Loss | 0.0653 |
| L1 Loss | 0.0326 |
| MSE Loss | **0.0680** |

Table 4. Effect of cycles

| Number of Cycles | Selection Ratio | Score↑ |
|---|---|---|
| 1 (2 epochs) | 10% | 0.0680 |
| 1 (2 epochs) | 20% | 0.0656 |
| 2 (2-2 epochs) | ~20% | 0.0844 |
| 3 (1-2-2 epochs) | ~30% | **0.0856** |

Table 5. Final submission score compared with other unlearning methods

| Model | Score↑ |
|---|---|
| Negrad | 0.0001 ($\pm$0.0001) |
| Fine-tune | 0.0464 ($\pm$0.0031) |
| Ours | 0.0935 ($\pm$0.0060) |
| Ours, best | **0.1024** |

# 7th place solution - Jiaxi Sun

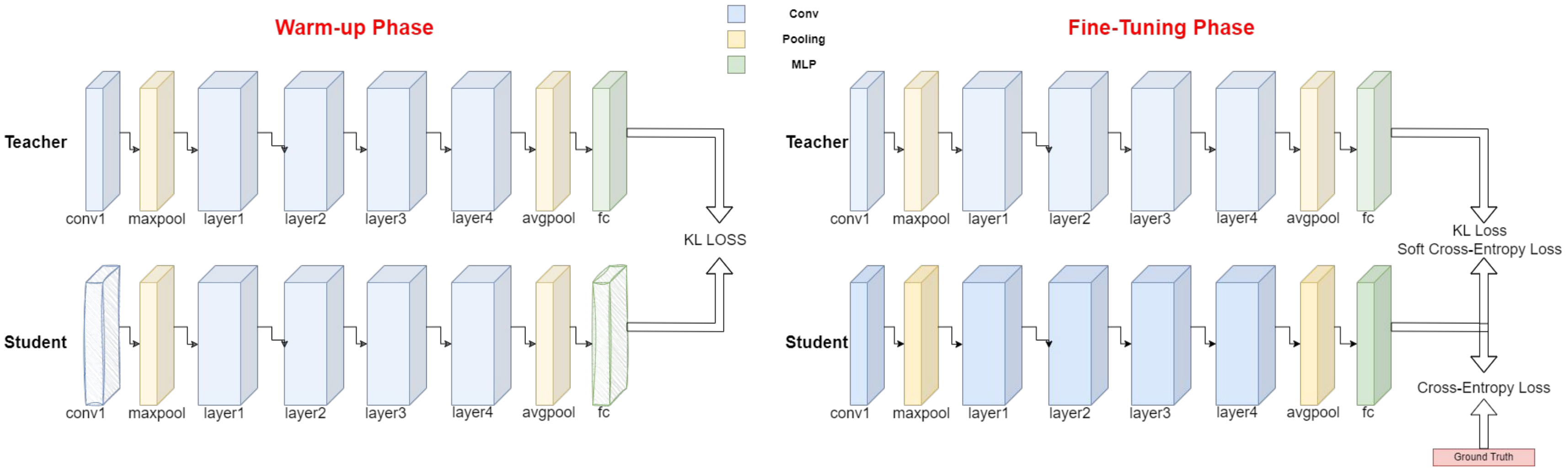
Solution that only makes use of retain set

1. Reset parameters of last layer

1. Randomly selecting N=9 layers from the network and add noise
   a. Adding noise helps the network 'forget' the information it has learned, and the randomness of the layer selection contributes to enhancing the model's diversity.

1. Fine-tune all network layers

# 6th place solution - Algorithmic Amnesiacs



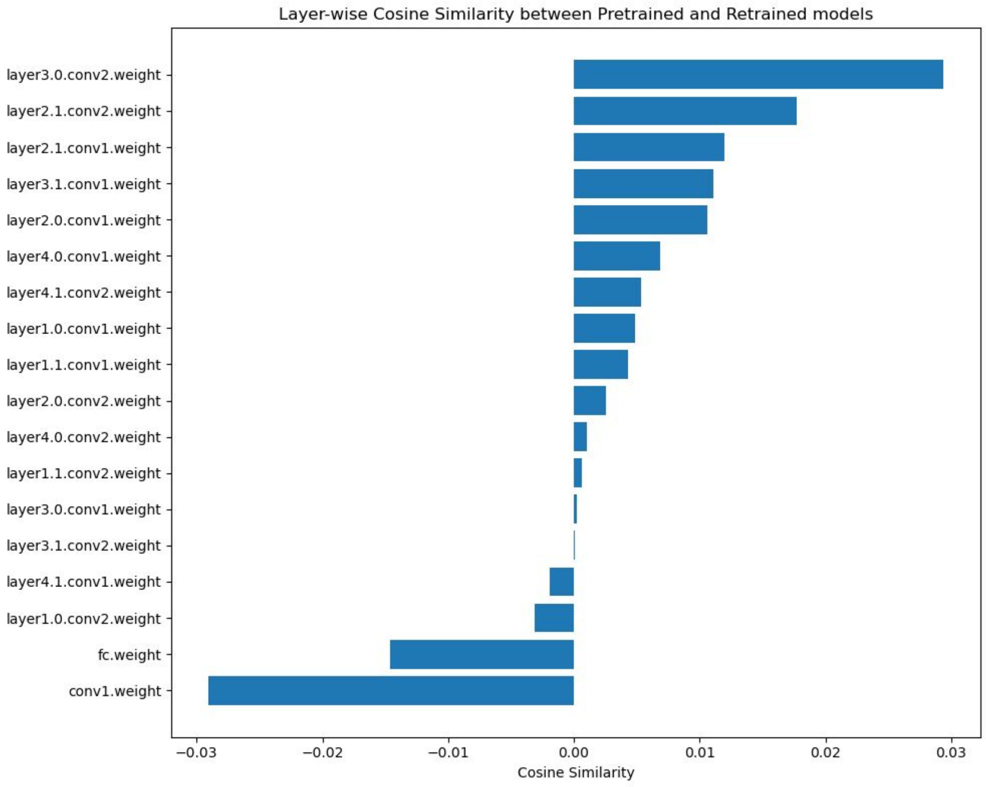

1. Reset first and last layer of the original model.
2. Warm-up phase employing knowledge distillation
3. Fine-tuning phase.

# 6th place solution - Algorithmic Amnesiacs

## Reset first and last layers:

- First layer significantly influences the rest of the model layers and the last layer determines the model's final output distribution.

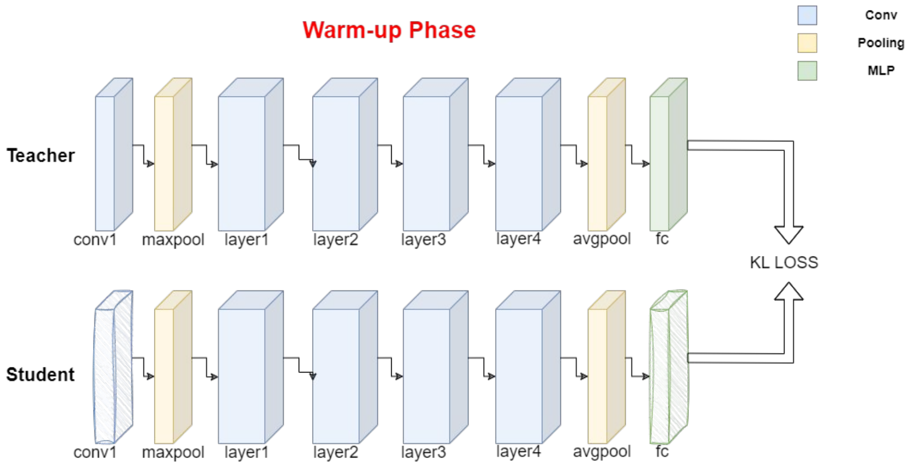
- On CIFAR-10: these two layers exhibited the most negative cosine similarity between model weights trained on the full training set and models trained from scratch on a smaller subset (i.e., the retain set).



# 6th place solution - Algorithmic Amnesiacs

## Warm-up phase

Minimize KL divergence between the outputs of the original pre-trained model (teacher) and the reinitialized model (student) **on the validation set**.



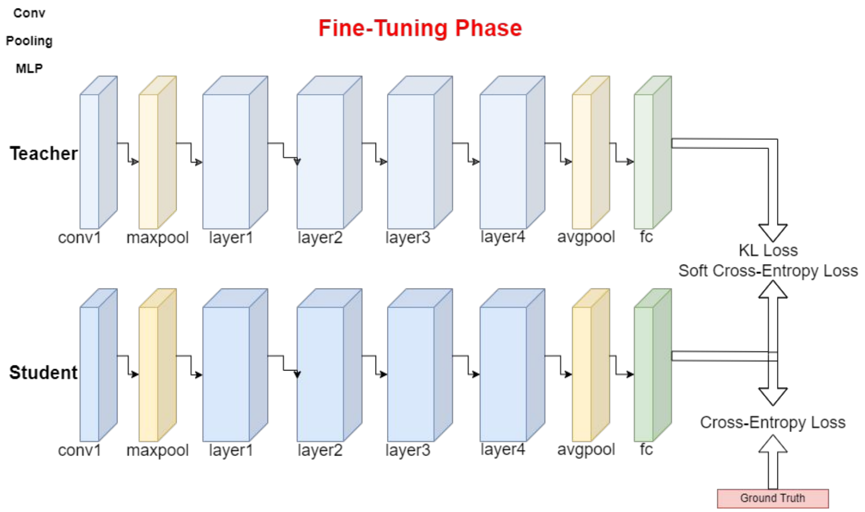
# 6th place solution - Algorithmic Amnesiacs

## Fine-tuning using 3 losses

**Cross-entropy** for model's accuracy using hard labels on retain set.

**Soft cross-entropy** for predictions of the student model with soft labels from the teacher model.

**KL divergence** combined with the soft cross-entropy facilitates rapid knowledge transfer and broader information capture.



# 5th place solution - toshi_k & marvelworld

## Summary

Our solution is the ensemble of two approaches:
(1) Retraining from transposed weights
(2) Fine-tuning with pseudo-labels.

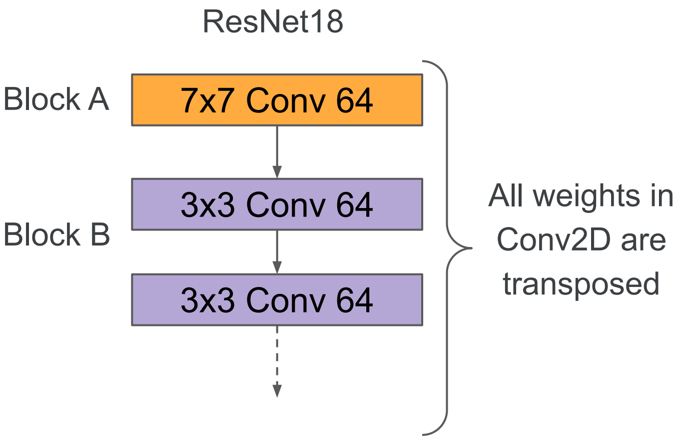
Our solution is built upon two distinctive approaches, contributing to the stability of our solution in the private LB.

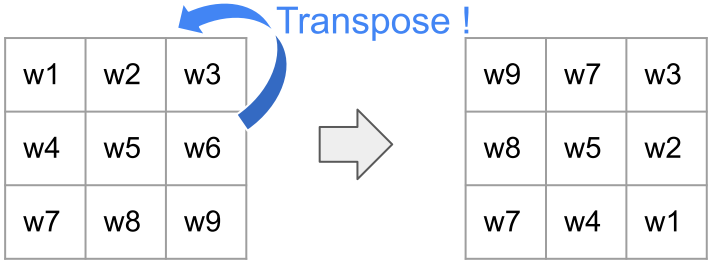
| (1) Retraining from transposed weights | (2) Fine-tune with pseudo-labels | Public LB | Private LB |
|---|---|---|---|
| 512 models | 0 models | 0.0720386947 | - |
| 0 models | 512 models | 0.0707241647 | - |
| 246 models | 266 models | - | **0.0785184178** |
| 266 models | 246 models | - | 0.0756313425 |

# 5th place solution - toshi_k & marvelworld

## (1) Retraining from transposed weights

- This part retrains the model using a modified version of the original model.
- In this modification, all weights in Conv2D are transposed. This process helps in forgetting samples in the forget-set, enabling the reuse of valuable features from the original model.




```
for module in local_model.modules():
    if isinstance(module, torch.nn.modules.conv.Conv2d):
        module.weight = torch.nn.Parameter(module.weight.swapaxes(2, 3))
```

The modification is carried out simply like this.

# 5th place solution - toshi_k & marvelworld

## (2) Fine-tuning with pseudo-labels

- This part reproduces behavior (errors) on the forget data with pseudo-labels from two functions.

**Algorithm 1** Unlearning with pseudo-labels

**Input:** forget data : $(x_f, y_f) \in D_F$
retrain data : $(x_r, y_r) \in D_R$
pretrained model : $f_\theta^p$
simple finetuning model : $f_\theta^t = f_\theta^p(x_r) \rightarrow \hat{y}^{pr} \rightarrow Loss(\hat{y}^{pr}, y^r)$
simple scratch model : $f_\theta^s = f_\theta(x_r) \rightarrow \hat{y}^r \rightarrow Loss(\hat{y}^r, y^r)$
confidence threshold : $T$

**function** INCORRECT DIRECTION PSEUDO-LABELS
  **for** $x_f \in D_F$ **do**
    **if** $f_\theta^p(x_f) = y^f$ **and** $f_\theta^p(x_f) \neq f_\theta^t(x_f)$ **then**
      $D_P \ni \{x_f, \hat{y}^{tf}\}$
    **end if**
  **end for**
  **return** $D_P$
**end function**

**function** HIGH CONFIDENCE INCORRECT PSEUDO-LABELS
  **for** $x_f \in D_F$ **do**
    **if** $Entropy(f_\theta^s(x_f)) < T$ **and** $f_\theta^s(x_f) \neq y^f$ **then**
      $D_P \ni \{x_f, \hat{y}^{sf}\}$
    **end if**
  **end for**
  **return** $D_P$
**end function**

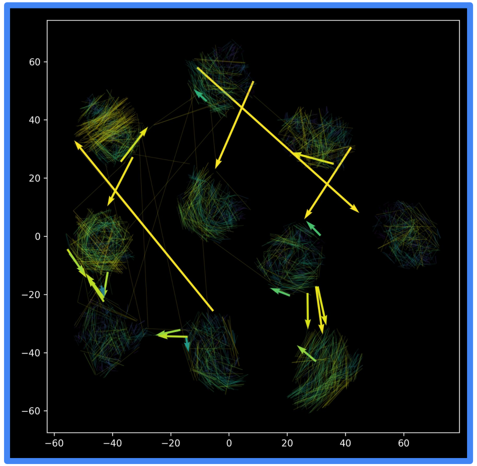



**Figure 1:** Inference shifts between the pretrained model and the fine-tuned model, showcasing significant shifts in the incorrect direction.

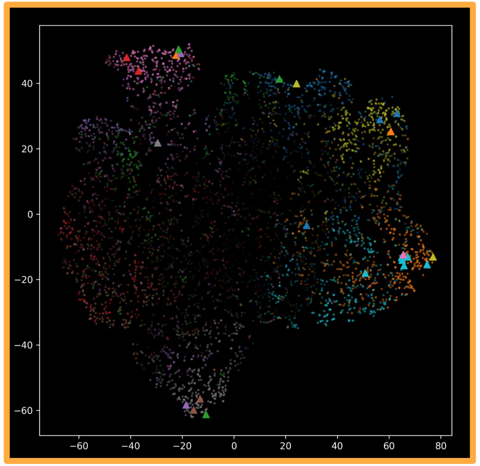



**Figure 2:** Inference results of the scratch model. Triangle-up marker indicates high confidence but they are making incorrect inferences.

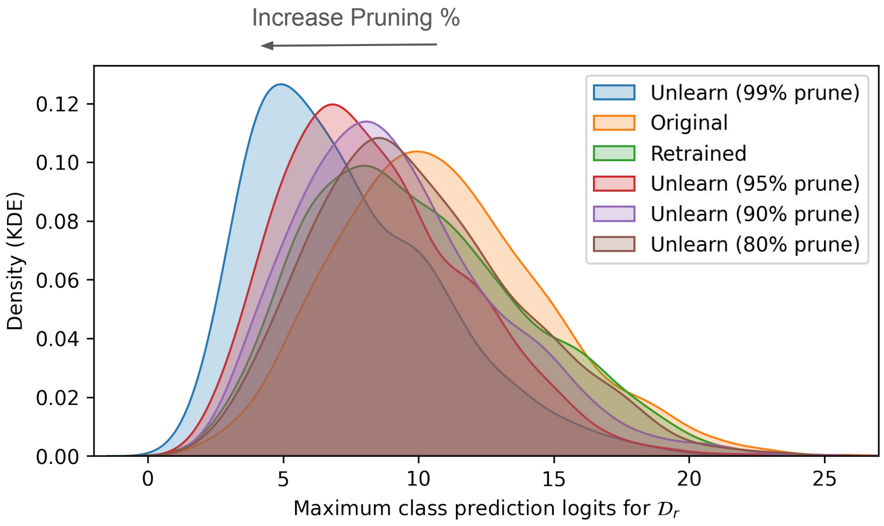
# 4th place solution - Sebastian Oleszko

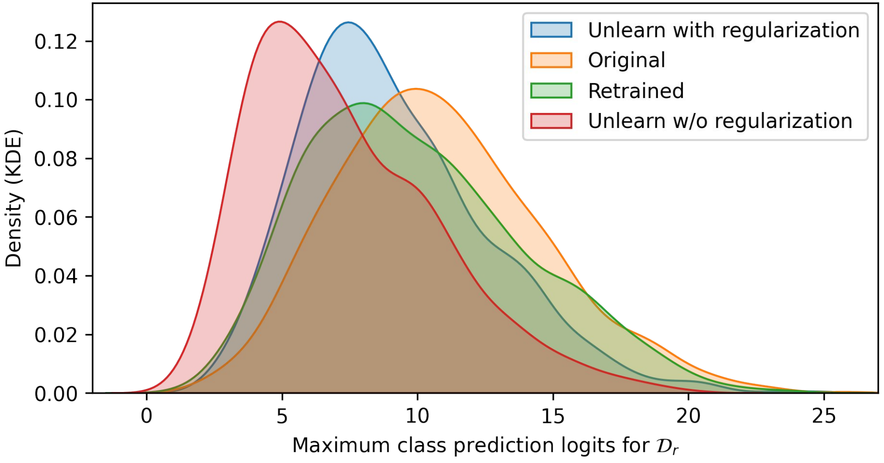
1. Re-initializing/pruning 99% of parameters based on L1-norm (Unstructured)
    - Weights: Pytorch default initialization
    - Biases: Set to zero (prune)
2. Fine-tune on retain dataset
    - Regularize using entropy
    - Cross entropy class weights as $N_c^{-0.1}$

$$\min_{w} \underbrace{\sum_{(x_r, y_r) \in \mathcal{D}_r} H(y_r, f(x_r; w))}_{\text{Cross-entropy}} + \underbrace{\sum_{x_r \in \mathcal{D}_r} \left(H(f(x_r; w)) - H(f(x_r; \overset{\text{Initial weights}}{\downarrow}{w^o}))\right)^2}_{\text{MSE of entropy}}$$

# 4th place solution - Sebastian Oleszko

- CIFAR-10 experiment
- Impact of most important hyperparameters: Learning rate/epochs and pruning percentage
- Effect of including entropy regularization
- Tuning on public submission scores



# 4th place solution - Sebastian Oleszko

## Entropy-based regularization

Helps to achieve a more similar prediction distribution/confidence.

## Unlearning through pruning/re-initialization

Effective as unlearning technique. Most of the performance is retainable even with high pruning percentage.

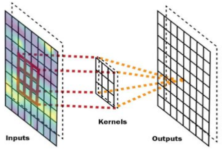
## Concluding thoughts

- Hyperparameter tuning is **very** important to achieve high scores
- Final submission was only fine-tuned for 3.2 epochs - maybe not optimal

# 3rd place solution - Seif Eddine Achour

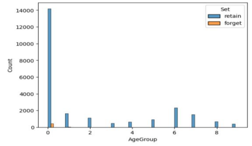
## Competition approach: vision confusion - reconstitution

### Vision Confusion




Details forgotten but the general idea was retained

### Class Imbalance

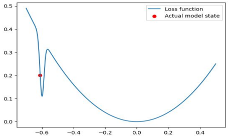



(b) Retain and forget histograms

Weighted cross entropy

Vision reconstitution on the Retain Set through 3 epochs

### Hard Differentiability




Slight confusion & a final stabilisating epoch

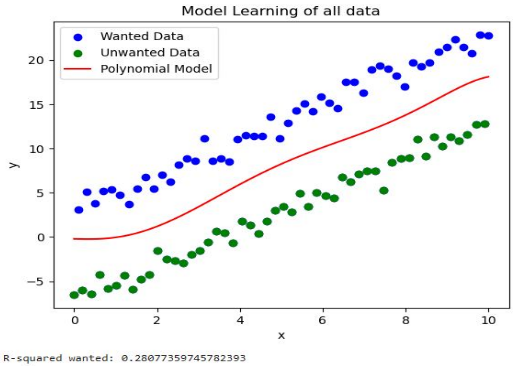
Model Unlearned!

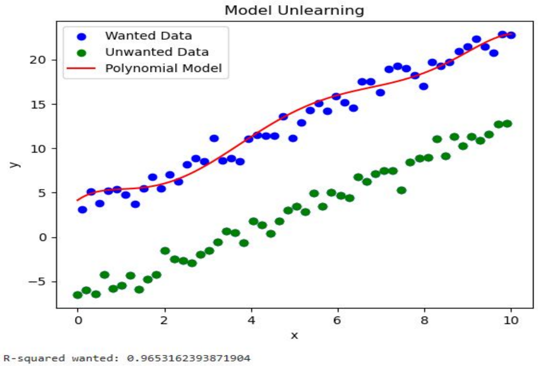
The confusion process is instant. The whole computation is dedicated to the vision reconstitution (Time Efficient).

# 3rd place solution - Seif Eddine Achour

## Paper approach: Loss landscape adjuster

**Regression**



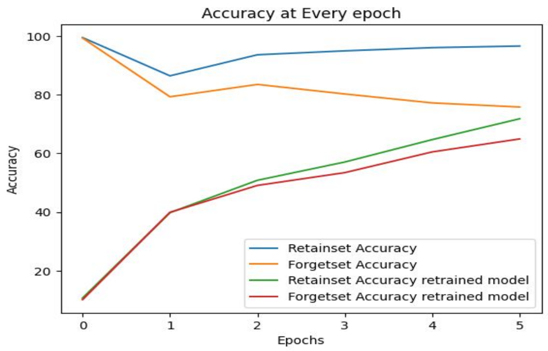

**The model forgot totally about the unwanted data despite its big size**

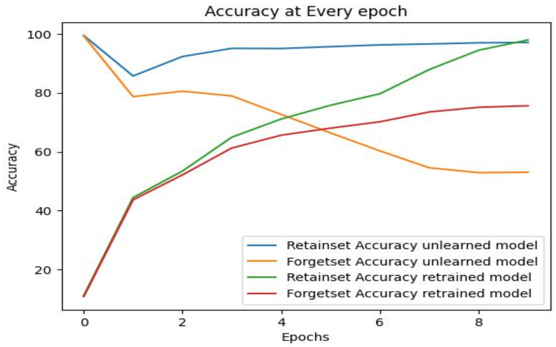
**Original r-squared = 0.28 vs Unlearned r-squared = 0.97!**

# 3rd place solution - Seif Eddine Achour

## Paper approach: Loss landscape adjuster

### Classification




- **Good results without considering the Forget Set (1st approach)**

➡ **The proper use of the Forget Set will certainly improve results**

- **The retrained model is not always the ideal one**

➡ **-1% accuracy on Retain Set lead to -24% of accuracy on Forget Set**

➡ **The metric which check the similitude between the unlearned and retrained model is not that representative for the unlearning performance**

# 2nd place solution - [kookmin Univ] LD&BGW&KJH

**Gradient-based re-initialization method**
We assumed that if the gradients of the weights in the model, specifically in the retain set and forget set, are similar, it becomes challenging to forget information from the forget set during the retraining of the retain set.

proposed gradient-based re-initialization method for unlearning consists of three main steps:

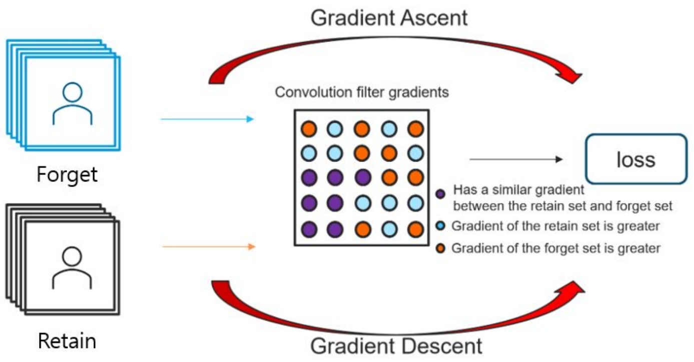
1. **Gradient Collection**:
   Gradient information is collected from the forget set and the retain set.

2. **Weight initialization**:
   Based on the gradient information collected in the first step, a percentage of the convolution filter weights are re-initialized.

3. **Retraining**:
   The model is retrained with the retain set. The learning rate for the Uninitialized weights uses 1/10 of the base learning rates.

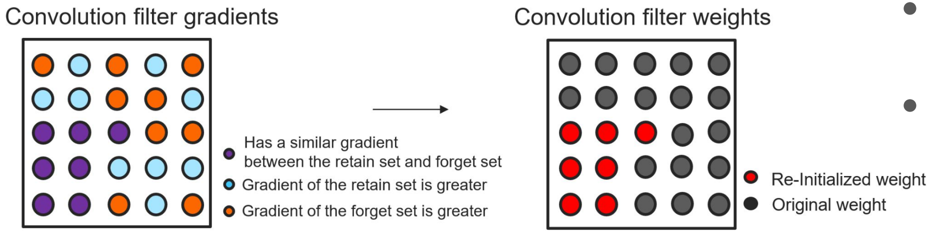
# 2nd place solution - [kookmin Univ] LD&BGW&KJH

1. Gradient Collection




- Collect gradients of forget set using gradient ascent
- Collect gradients of retain set using gradient descent
- Random sampling was used from the retain set to match the number of samples in the forget set
- In short, this is simply subtracting forget set's gradient from the retain set's gradient

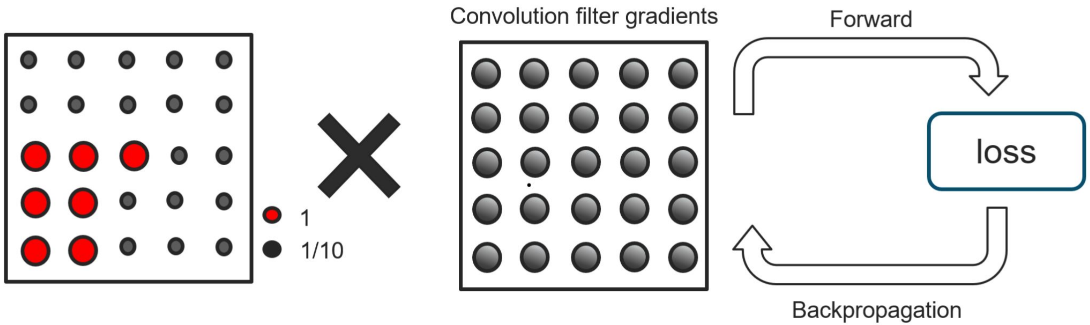
2. Weight initialization




- Based on the gradient information a percentage of the convolution filter weights are re-initialized
- Our best method re-initialized 30% of convolution filter weights

# 2nd place solution - [kookmin Univ] LD&BGW&KJH

3. Retraining




- Re-initialized Model is trained using the retain set
- Learning rate for the uninitialized weights uses 1/10 of the base learning rates (accomplished by scaling the gradient of uninitialized weight)
- Used a linear decay learning rate scheduler with a few warmup epoch
    - Consistently produces better results than other learning rate schedulers
    - Used warmup epoch of 3
      (0.00033 to 0.001 in the first 3 epochs, and then linearly decreases from 0.001 to 0.00033 in the last 2 epochs)

# 1st place solution - fanchuan

1. **Forget phase**: minimize KL-divergence between output logits and a uniform pseudo label on forget set.

1. **Adversarial fine-tuning phase**. Alternate between "forget" and "retain" rounds:

Forget round: Maximize dissimilarity between logits of forget and retain set

$$l_i = -\frac{1}{bactshsize2} \sum_{t=0}^{batch_2} log(\frac{e^{sim(x_i, y_t)/\tau}}{\sum_{j=0}^{batch_2} e^{sim(x_i, y_j)/\tau}})$$

$$L_{forget} = \frac{1}{batchsize_1} \sum_{i=0}^{batch_1} l_i$$

Retain: original loss (cross entropy) on retain set.

*Trick*: Increase batch size from 64 to 258 to be able to perform more epochs (6 -> 8)