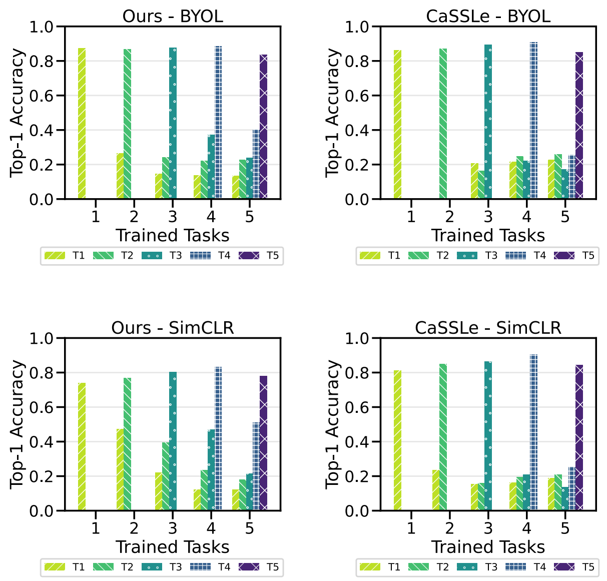

Figure 10. **Detailed breakdown of performance over tasks on Imagenet-100**. Fine-grained accuracy for every additional task across `Kaizen` and CaSSLe, with a fixed SSL backbone.

A. Additional results

In addition to the results discussed in section 5.4, Figure 11 displays the performance of the models on each individual task in the continual learning process on CIFAR-100, using different self-supervised learning methods (BYOL [16], SimCLR [8], ViCReg [1]) for training and knowledge distillation, and different distillation strategies (`Kaizen`, *No Distill* and CaSSLe). We can again observe that our proposal is able to retain knowledge better compared to other methods. The *No Distill* retains the least amount of knowledge where the performance drops to almost zero after learning a new task. We can make similar observations on ImageNet-100 (see Figure 10), where our method is generally more able to mitigate forgetting, although this effect is more apparent when using SimCLR than BYOL.

B. Hyperparameters

In our experiments, we retained the hyperparameters for each self-supervised learning method (BYOL [16], SimCLR [8], ViCReg [1]), MoCoV2+ [9]) as they were originally proposed in their corresponding works. This was shown to reduce interference [14] and used to make sure that the comparison is not a result of hyperparameter tuning. In the loss function (Equation 1), it is possible to use a different weight for each loss function term, to enhance or reduce the supervisory signal from different objectives. A weighting factor of 2 was empirically chosen for the knowledge distillation loss for the classifier while others are kept as 1. Since the replay dataset is much smaller in size compared to the dataset of the current task, we ensure that each batch that the models are trained on contains at least 32 samples from the replay dataset. The replay dataset is reset as many times as necessary in order to match the number of batches in the data of the current task.

Algorithm 1 Algorithm of `Kaizen`.

```
# aug_f: stochastic augmentation function
# f_o: Current Feature Extractor
# f_t: Momentum Feature Extractor (if applicable)
# f_p: Previous Feature Extractor
# h_o: Predictor for Knowledge Distillation
# h_t: Predictor for SSL
# g_t: Current Classifier
# g_p: Previous Classifier
# loss_ssl: self-supervised learning loss
# loss_ce: cross-entropy loss

def train_step(x, y):
    # augmented views of input
    x1, x2 = aug_f(x), aug_f(x)

    # pass through feature extractors
    z_o = f_o(x1)
    z_t = f_t(x2)
    z_p = f_p(x1)

    # pass embeddings through classifiers
    c_t = g_t(z_t.detach()) # detach stops gradients
         backpropagation
    c_p = g_p(z_p)

    # pass embeddings through predictors
    p_o = h_o(z_o)
    p_t = h_t(z_o)

    # knowledge distillation for feature extractor
    kd_fe = loss_ssl(p_o, z_p.detach())

    # knowledge distillation for classifier
    kd_c = loss_ce(c_t, c_p.detach())

    # supervised training for current task
    ct_c = loss_ce(c_t, y)

    # SSL training
    ct_fe = loss_ssl(p_t, z_t)

    # Overall loss
    loss = kd_fe + kd_c + ct_c + ct_fe

    return loss
```

C. Algorithm

The algorithm of our training pipeline is provided in this section (Algorithm 1) to accompany the descriptions given in Section 3.2 and illustrated in Figure 3 using a PyTorch-like syntax, which outlines the implementation of `Kaizen`.

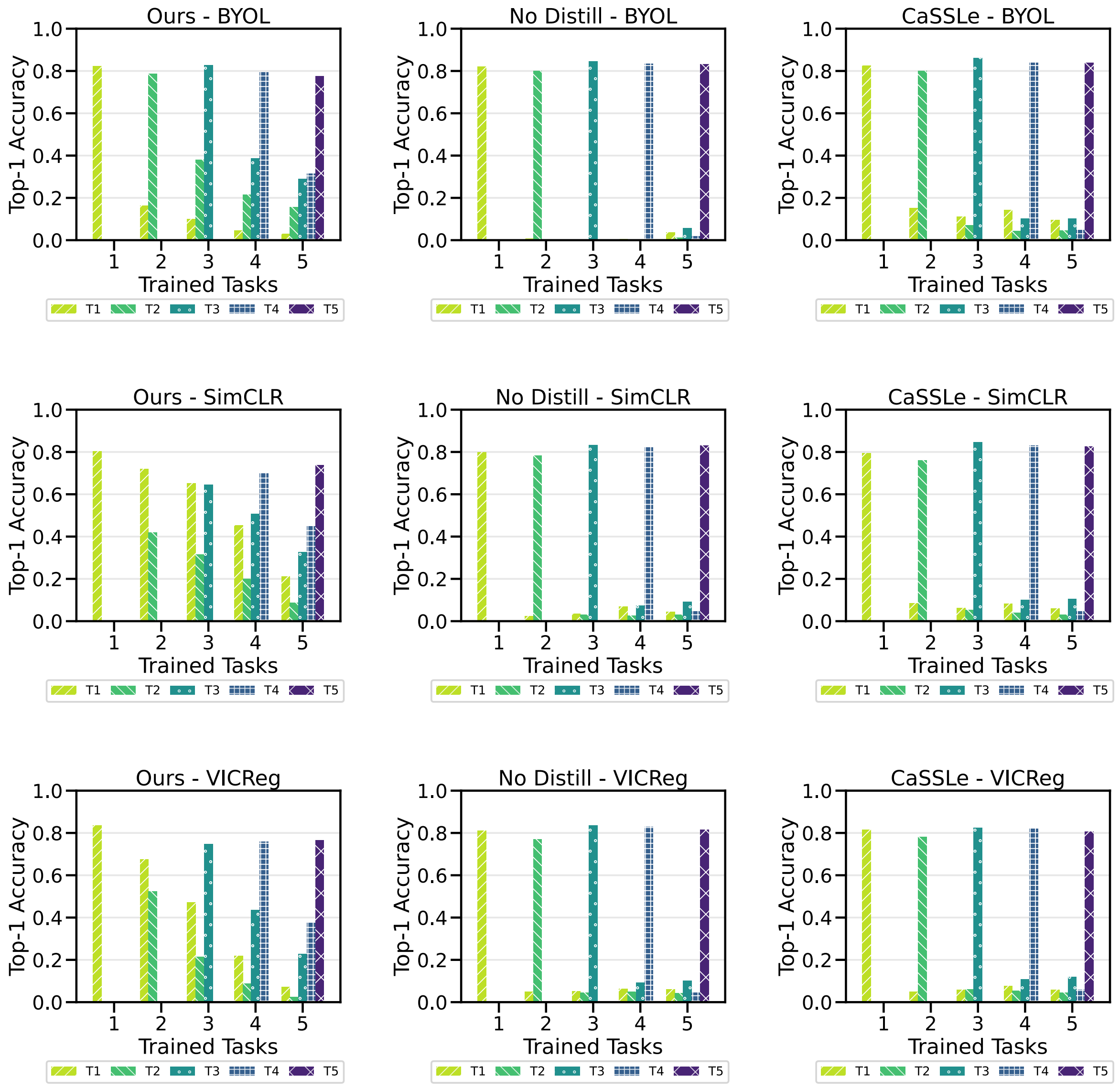

Figure 11. **Detailed breakdown of performance over tasks on CIFAR-100**. Fine-grained accuracy for every additional task across `Kaizen` and CaSSLe, with a fixed SSL backbone.