A Supplementary Material

First, we illustrate our whole training and inference steps in Figure **??**. Next, we illustrate an example of our genetic search approach for three classes in Figure 3, and we also show some mutation examples in Figure 4. In section A.1 we include further analysis for the manifold intrusion problem. Next, we show some qualitative examples for the Weakly Supervised Object Localization and Object Detection experiment using the Class Activation Mapping (CAM) in A.2. Additional ablation studies for PatchMix varying architectures, the supervision effect of the patches, and the grid size (number of P patches) are shown in section 5.5. We also include results when varying the genetic search fitness function and training criterion for our guided PatchMix approach in section 5.6. We show in-depth results of the search performance of our genetic algorithm for CIFAR-10 in section A.5. Finally, we discuss the computational overhead of PatchMix compared to other similar interpolation approaches in section A.7.

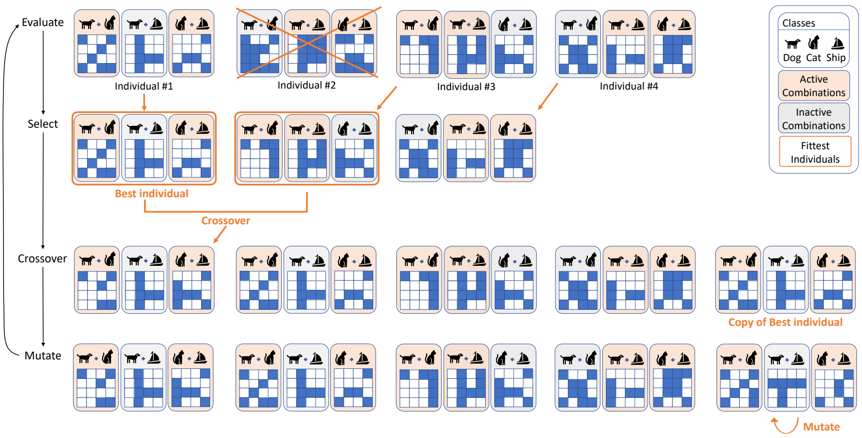

Figure 3: Overview of our genetic search over patch combination strategies for a three-way dog, cat, ship classifier. First, we randomly sample a population of four individuals containing a set of Grid Mask configurations for all combinations. Then we evaluate them using our fitness function (i.e., our PatchMix algorithm), and the individuals with the less accurate predictions get selected. We randomly pick two individuals to crossover and create a new individual (i.e., offspring), and a copy of the best individual is mutated. The next cycle begins with the evaluation of the current pool of individuals, and the process repeats.

A.1 Manifold Intrusion Analysis.

We show in Figure 5 the effect in the decision boundary for a three-way classifier on synthetic data when using different interpolations such as Mixup, Cutout, and our proposed PatchMix.

The main advantages of PatchMix include preventing the over-sampling of synthetic data that is prone to suffer the *manifold intrusion* problem during training. We train a two-layer neural network to classify a toy dataset with three linearly separable classes. Since we are using only two features per class, the Cutmix and Random PatchMix interpolations behave similarly (for both cases, the random masks have only two choices: $0|0$, $1|0$, $0|1$ or $1|1$).

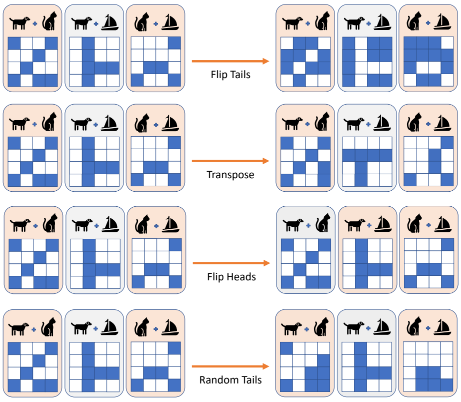

Figure 4: Mutation examples. The original individual is in the left and its mutated version is in the right. The *flip tails* operation exchange all the values from the active mask configurations. The *transpose* operation switches the row and column indices of each patch. The *flip heads* operation randomly exchange the active configurations. The *random tails* operation randomly activate and deactivate some values of the active masks.

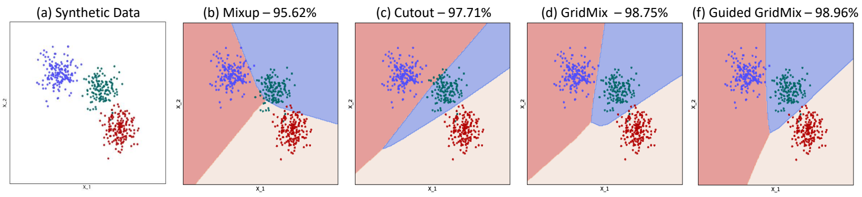

Figure 5: Effect of different data interpolation techniques on the decision boundary for a classification task.

While all interpolation techniques smooth the decision boundaries, Mixup and Cutout suffer from the *manifold intrusion* problem, hurting the model. Our guided version of PatchMix induces a better decision boundary by adding predefined combinations based on difficult mixed samples.

A.2 Weakly Supervised Object Localization and Object Detection.

Figure 6 shows some results for the weakly supervised object localization task on CUB-200-2011 dataset trained on ResNet-50. We show qualitative results when using the CAM baseline, Mixup, CutMix and PatchMix. The ground truth bounding boxes are shown in red and the predicted bounding boxes are shown in light green. We use the Class Activation Mapping (CAM) to extract the attention maps, and then we compute the maximal box accuracy, which is the bounding box accuracy and the Intersection over Union (IoU) of the proposed boxes.

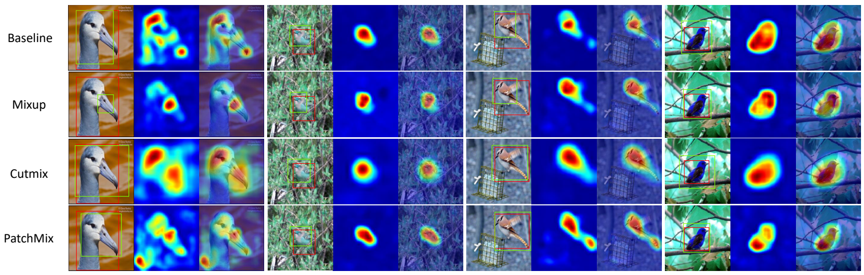

Figure 6: Weakly supervised object localization task on CUB-200-2011 dataset trained on ResNet-50. The ground truth bounding boxes are shown in red and the predicted bounding boxes are shown in light green.

A.3 Analysis on Search for Guided Pairs & Masks

Figure 8 shows the evolution of our fitness function when searching for the grid mask configurations on CIFAR-100 over 250 generations. Since we are looking for the most informative samples, we want to find the class combinations and grid configurations for which a pretrained Random PatchMix struggles the most. Hence, our criteria for fitness allows the discovery of individuals that contribute with more information as explained in our method section. We observe that during the search process, some class combinations get discarded systematically in the search process. For instance, the model initially selects the pairs (*plain*, *seal*), (*chimpanzee*, *mushroom*), but those are discarded after 50 generations and more informative combinations are selected. After 100 generations, the model has discovered many of the class combinations that it will use, such as (*chimpanzee*, *raccoon*), (*road*, *tractor*), and (*sea*, *shark*).

Evolutionary search has been used in previous work for neural architecture search (NAS) where a space of possible network designs is explored, and can be extended for data augmentation where a combinatorial space of image transformations are explored. In both cases, the bottleneck is to evaluate each configuration – as it requires training a deep neural network to assess these choices. Our work goes beyond that as we are able to entirely bypass any amount of training. Our idea of using a PatchMix model pretrained on random configurations to define a fitness criteria allows for this to happen.

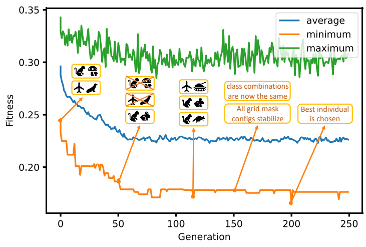

Figure 7: Performance of our genetic search after 250 generations on CIFAR-100. The y axis shows the top-1 accuracy of the population evaluated on our fitness function. We highlight some class combinations included and excluded every 50 generations.

A.4 Additional Ablation Studies.

In this section we show additional results when varying the grid size (number of P patches), the model architecture and whether the image or patch supervision are used in the training process. Table 8 shows that using the image level supervision along with the patch level supervision consistently outperforms all other variations. Furthermore, in all architectures a grid size of 4×4 outperforms other grid size possible selections.

	Grid	Image-level loss L_O	Patch-level loss L_P	Top-1 Acc
PreActResNet-56	2×2	✓	✓	**94.53**
	2×2	✓	✗	93.78
	2×2	✗	✓	94.03
	4×4	✓	✓	**95.28**
	4×4	✓	✗	94.97
	4×4	✗	✓	94.73
	8×8	✓	✓	**94.08**
	8×8	✓	✗	92.84
	8×8	✗	✓	92.57
PreActResNet-164	2×2	✓	✓	**94.55**
	2×2	✓	✗	94.27
	2×2	✗	✓	94.53
	4×4	✓	✓	**96.32**
	4×4	✓	✗	94.56
	4×4	✗	✓	94.37
	8×8	✓	✓	**94.60**
	8×8	✓	✗	94.03
	8×8	✗	✓	93.86

Table 8: Ablation analysis: Top-1 accuracy on CIFAR-10 when varying the grid size, and the effect of using image and patch level supervision using different network backbones.

A.5 Additional Search Performance

Figure 8 shows how the evolution of our fitness function when searching for the grid mask configurations on CIFAR-10 over 250 generations. Since we are looking for the most difficult samples, we want to find the class combinations and grid configurations for which a pretrained model using PatchMix with random sampling struggles the most. Hence our criteria for fitness is individuals that score the lowest under this criteria as explained in our method section. We observe that when searching the space, even the score of the individuals with the best top-1 accuracy drops, which is expected in our setup. We also observe that during the search process, some class combinations get discarded systematically in the search process. For instance, the model initially selects the pair (*automobile*, *bird*), and (*airplaine*, *deer*), but those pairs are discarded after 50 generations and more informative combinations such as (*automobile*, *airplane*) are selected. After 100 generations, the model has already discovered many of the class combinations that it will use such as (*automobile*, *airplane*), (*dog*, *horse*), (*dog*, *cat*), and after 150 epochs converges to a set of categories and the grid mask patterns stabilize as well. Finally, at around epoch 200, the best individual containing the best pairs and mask patterns is chosen.

A.6 Impact of Selected Number of Configurations

We also show in Table 9 the impact of using different amounts of active combinations when evaluating our approach with CIFAR-100. In general, the search algorithm performs better when setting a limited amount of active combinations N equal to the number of classes. We also observe that the best results are achieved when using only the discovered class combinations. This also supports our rationale about the positive gains of the model when training using the configurations discovered by our genetic algorithm. In general, our evolutionary formulation is effective and yields the best results.

A.7 Computational Overhead

Random PatchMix does not add a significant overhead -our mask is randomly sampled-, and the additional patch-level branches make the model $\approx 20\%$ larger; i.e., a modified model did not surpass any of the available GPU capacity during our experiments. For our guided version, the overhead depends on the number of classes and active combinations; i.e., for CIFAR10, using a set of 10 parallel processes to assess the fitness criterion, it would take ≈ 4 hours to find the optimum combination. This can be further decreased for at least $\approx 75\%$ by using a subset of the val. set randomly sampled from the whole val. set, shrinking the initial population, and reducing the number of generations (we found that the genetic algorithm stabilizes around the 60th generation, but we continue searching for 250 generations). During test time, Guided-PatchMix, does not add any significant overhead compared to other similar methods.

N Comb	Allow Same Class Pairs?	
	Yes	No
200	77.00	**78.53**
300	76.61	77.55
1000	64.98	65.54

Table 9: Top-1 accuracy on CIFAR-100 when varying the number of active class combinations. We also show the effect of using the same class pairs. We use PreAct-ResNet-164 as the backbone network architecture.

Figure 9 shows the final top 10 class combinations found by the genetic algorithm on CIFAR-10. When training using the guided approach, the combination of these classes are more likely to be sampled. The evolutionary process selects many pairs that are semantically close such as (*dog*, *cat*), (*dog*, *horse*), (*ship*, *truck*), or (*ship*, *airplane*).

A.8 Additional Implementation Details

In this section we present the pseudo-code to train our random and guided steps four our PatchMix approach. We first train a network using Algorithm 1 in which we divide the last convolutional layer of our network to output the same amount of patches like the ones defined

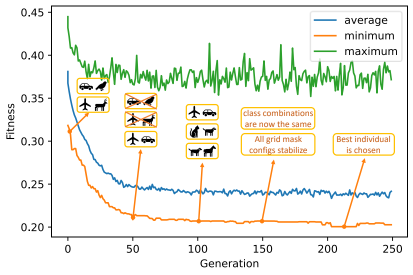

Figure 8: Performance of our genetic search after 250 generations on CIFAR-10. The y axis shows the top-1 accuracy of the population evaluated on our fitness function, which is a network trained with PatchMix and random sampling. For this example, the architecture of the network is PreAct-ResNet-164. We highlight some class combinations included and excluded every 50 generations.

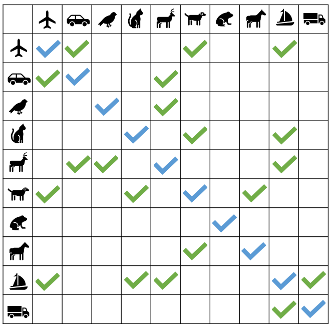

Figure 9: Class combinations. The green checks correspond to the class combinations found by our search algorithm. The blue checks correspond to the classes we force to always appear in the combinations.

in the input. For example, the convolutional layer of the latest block in a ResNet50 will output a tensor of shape $[N, C_{out}, H_{out}, W_{out}]$; if the input corresponds to images of $224x224x3$ and we divide our image in 16 patches $(P = 4)$, $C_{out} = 2048$ denotes the number of channels and $H_{out} = 28$, $W_{out} = 28$ are the height and width of this output. Thus we can also divide this output into a grid-like layer in which each patch would be of size $7x7$. We then apply the average pooling over each of these patches and finally the linear transformation to output the probability distribution of each patch prediction. This network also outputs a separate branch that takes into account the whole convolutional layer to predict the probability distribution of each input image as a whole. We then use genetic search, which we define in Algorithm 2 to find the best masks $M_{i,j}$ and category pairs (c_i, c_j) that correspond to each of the discovered class combinations. We then use $M_{i,j}$ to augment the training samples based on the class combinations (c_i, c_j) discovered by our genetic search.

Algorithm 1 Pseudo-code of our first step: Random PatchMix

```
Require: $P$
for (x, y) in (data_loader) do
    index = torch.randperm(batch_size)
    lam = random.beta(1, 1)
    quad_mask_vals = random.beta(1, 1, size=[P, P]).round()
    quadrant_size = int(image_size/P)
    result = []
    for r in range(P) do
        rows = full((quadrant_size,quadrant_size), quad_mask_vals[r,0])
        for c in range(1, P) do
            row_column = full((quadrant_size,quadrant_size), quad_mask_vals[r,c])
            rows = concatenate((rows, row_column), axis=1)
        result.append(rows)
    image_mask = result.reshape(image_size,image_size)
    layer_mask = result.reshape(layer_size,layer_size)
    mixed_x = zeros((x.shape))
    mixed_x = x*image_mask + x[index]*(1-image_mask)
    new_y = []
    layer_mask = list(layer_mask.flatten())
    for m in layer_mask do
        if m == 0 then
            new_y.append(y[index])
        else: new_y.append(y)
    lam = layer_mask.count(1)/len(layer_mask)
    patch_output, image_output = model(mixed_x)
    $L_P$ = patch_level_criterion(patch_output, new_y, class_criterion, grid_range = $P^2$)
    $L_O$ = image_level_criterion(class_criterion, image_output, y, y[index], lam)
    loss = ($L_O + L_P$) / 2
    update(model)
end
```

Algorithm 2 Pseudo-code of our Evolutionary Search

```
head = get_array_activations()
tail = get_array_configurations()
population = []
for i in range(total_pop) do
    tmp_indiv = zeros(head)
    selected_head = random.sample(range(index_head), new_comb_count)
    tmp_indiv[selected_head] = 1
    tmp_indiv[default_head] = 1
    tmp_indiv_tail = random.randint(0, 2, tail)
    individual = list(concatenate((tmp_indiv, tmp_indiv_tail)))
    individual = creator(individual)
    population.append(individual)
toolbox = DEAP.Toolbox()
toolbox.register("population", load_population(), population_size)
toolbox.register("evaluate", eval_population())
toolbox.register("mate", crossover(), prob=0.50)
toolbox.register("mutate", mut_ops(), prob=0.30)
toolbox.register("select", toolbox.selTournament, tournsize=3)
for gen in range(0, args.generations) do
    offspring = toolbox.select(population, len(population))
    offspring = algorithms.varAnd(offspring, toolbox, cxpb, mutpb)
    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses) do
        ind.fitness.values = fit
    if halloffame is not None then
        halloffame.update(offspring)
    population[:] = offspring
end
```