Supplementary Material for
PEFAT: Boosting Semi-supervised Medical Image Classification via Pseudo-loss Estimation and Feature Adversarial Training

Qingjie Zeng∗ Yutong Xie∗∥ Zilin Lu† Yong Xia††
1 School of Computer Science and Engineering, Northwestern Polytechnical University, China
2 The University of Adelaide, Australia
maxwell@mail.nwpu.edu.cn, yutong.xie678@gmail.com, luzl@mail.nwpu.edu.cn, yxia@nwpu.edu.cn

1. More Description of FAT and VAT

Algorithm 1 and Algorithm 2 present the pseudo-code of FAT and VAT in the Pytorch-like style, respectively. Here we can draw four observations: (1) VAT generates adversarial noise based on self-prediction, which may backfire when the original prediction is unsatisfied; (2) VAT simulates image-level adversarial noise, which may be learned by deep neural networks and thus show limited effects; (3) alternatively, in our work, we approximate the adversarial noise using two different views for the unselected low-confidence data. Therefore, we can mitigate the issue of undesirable adversarial noise caused by confirmation-biased self-prediction; (4) FAT injects feature-level adversarial noise, making the model adapt to a more difficult task, thereby boosts performance by better robustness.

2. Class-level AUC on Chest X-Ray14

Table 1 shows the performance comparison of class-level AUC on Chest X-Ray14 with label percentage of 20%. PEFAT achieves the best or second best class-level performance in 11 out of 14 categories. Specifically, compared to pseudo-labeling methods UPS and ACPL, our method has 0.81%~2.66% performance gain in averaged AUC even if the input image size is smaller. Beyond that, PEFAT outperforms consistency-based method SRC-MT and self-supervised learning-based method S2MTS2 by 3.35% and 1.52%, respectively.

3. Results on CIFAR-10 and CIFAR-100

We also conduct experiments on CIFAR-10 [2] and CIFAR-100 [2], following the same setup as [13]. Table 2 presents the results. We can find that PEFAT shows better performance on CIFAR-100 and is on par with other state-of-the-art methods on CIFAR-10. For most methods, there is little improvement on CIFAR-10 even if increasing the number of labeled data, and the main reason is that models generally make correct predictions on unlabeled data, making the results close to the fully supervised baseline. As for CIFAR-100, a relatively complicated dataset, PEFAT consistently achieves the best results under different settings, demonstrating its superiority of unlabeled data mining.

4. Discussion of Hyper-parameter η

In the proposed PEFAT, η is a hyper-parameter to balance the weight between two cross pseudo-losses (see Eq. (10)). As shown in Figure 1, we conduct a grid search of η ∈ {0.1, 0.3, 0.5, 0.7, 0.9} on NCT-CRC-HE dataset. We can find that PEFAT achieves the best performance when η = 0.5. This result indicates that it is the most effective to select trustworthy pseudo-labeled data when treating two cross pseudo-losses equally.

---

*Equal contribution. †Corresponding author.

Figure 1. Four evaluation metrics under different η, which are conducted on NCT-CRC-HE dataset.
5. Limitation

The proposed FAT utilizes loss backpropagation to approximate the feature-level adversarial noise, which may require higher demand for computing and storage on dense prediction tasks, *i.e.*, medical image segmentation. But for classification, the aforementioned cost can be almost ignored, since the classification head is lightweight.

References


Table 1. Performance of class-level AUC on Chest X-Ray14 dataset under the label percentage of 20%. Note that * denotes the methods employee DenseNet-169 as backbone with $384 \times 384$ input size, † means the methods use DenseNet-121 as backbone with $512 \times 512$ input size. Our method takes DenseNet-121 as backbone with input size of $224 \times 224$.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Atelectasis</td>
<td>75.75</td>
<td>75.12</td>
<td>75.38</td>
<td>77.09</td>
<td>78.57</td>
<td>79.53</td>
<td>79.77</td>
</tr>
<tr>
<td>Cardiomegaly</td>
<td>80.71</td>
<td>87.37</td>
<td>87.70</td>
<td>85.73</td>
<td>88.08</td>
<td>89.03</td>
<td>88.52</td>
</tr>
<tr>
<td>Effusion</td>
<td>79.87</td>
<td>80.81</td>
<td>81.58</td>
<td>81.35</td>
<td>82.87</td>
<td>83.56</td>
<td>86.30</td>
</tr>
<tr>
<td>Infiltration</td>
<td>69.16</td>
<td>70.67</td>
<td>70.40</td>
<td>70.82</td>
<td>70.68</td>
<td>71.40</td>
<td>70.98</td>
</tr>
<tr>
<td>Mass</td>
<td>74.40</td>
<td>77.72</td>
<td>78.03</td>
<td>81.82</td>
<td>82.57</td>
<td>82.49</td>
<td>81.56</td>
</tr>
<tr>
<td>Nodule</td>
<td>74.49</td>
<td>73.27</td>
<td>73.64</td>
<td>76.34</td>
<td>76.60</td>
<td>77.73</td>
<td>77.82</td>
</tr>
<tr>
<td>Pneumonia</td>
<td>69.55</td>
<td>69.17</td>
<td>69.27</td>
<td>70.96</td>
<td>72.25</td>
<td>73.86</td>
<td>74.18</td>
</tr>
<tr>
<td>Pneumothorax</td>
<td>84.70</td>
<td>85.63</td>
<td>86.12</td>
<td>85.86</td>
<td>86.55</td>
<td>86.95</td>
<td>87.80</td>
</tr>
<tr>
<td>Consolidation</td>
<td>71.85</td>
<td>72.51</td>
<td>73.11</td>
<td>74.35</td>
<td>75.47</td>
<td>75.50</td>
<td>78.92</td>
</tr>
<tr>
<td>Edema</td>
<td>81.61</td>
<td>82.72</td>
<td>82.94</td>
<td>83.56</td>
<td>84.83</td>
<td>84.95</td>
<td>87.43</td>
</tr>
<tr>
<td>Emphysema</td>
<td>89.75</td>
<td>88.16</td>
<td>88.98</td>
<td>91.00</td>
<td>91.88</td>
<td>93.36</td>
<td>92.86</td>
</tr>
<tr>
<td>Fibrosis</td>
<td>79.30</td>
<td>78.24</td>
<td>79.22</td>
<td>80.87</td>
<td>81.73</td>
<td>81.86</td>
<td>80.94</td>
</tr>
<tr>
<td>Pleural Thicken</td>
<td>73.46</td>
<td>74.43</td>
<td>75.63</td>
<td>75.55</td>
<td>76.86</td>
<td>77.60</td>
<td>75.42</td>
</tr>
<tr>
<td>Hernia</td>
<td>86.05</td>
<td>87.74</td>
<td>87.27</td>
<td>85.62</td>
<td>85.98</td>
<td>85.89</td>
<td>90.20</td>
</tr>
<tr>
<td>Mean</td>
<td>78.19</td>
<td>78.83</td>
<td>79.23</td>
<td>79.92</td>
<td>81.06</td>
<td>81.77</td>
<td>82.58</td>
</tr>
</tbody>
</table>

Table 2. Accuracy comparison (mean and std over 5 runs) with other state-of-the-art SSL methods on CIFAR-10 [2] and CIFAR-100 [2].

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR-10 250 labels</th>
<th>CIFAR-10 4000 labels</th>
<th>CIFAR-100 2500 labels</th>
<th>CIFAR-100 10000 labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT [9]</td>
<td>67.68 ± 2.30</td>
<td>90.81 ± 0.19</td>
<td>46.09 ± 0.57</td>
<td>64.17 ± 0.24</td>
</tr>
<tr>
<td>UDA [10]</td>
<td>91.18 ± 1.08</td>
<td>95.12 ± 0.18</td>
<td>66.87 ± 0.22</td>
<td>75.50 ± 0.25</td>
</tr>
<tr>
<td>FixMatch [8]</td>
<td>94.93 ± 0.65</td>
<td>95.74 ± 0.05</td>
<td>71.71 ± 0.11</td>
<td>77.40 ± 0.12</td>
</tr>
<tr>
<td>Dash [11]</td>
<td>95.44 ± 0.13</td>
<td>95.92 ± 0.06</td>
<td>72.82 ± 0.21</td>
<td>78.03 ± 0.14</td>
</tr>
<tr>
<td>CoMatch [3]</td>
<td>95.09 ± 0.33</td>
<td>95.44 ± 0.20</td>
<td>71.63 ± 0.35</td>
<td>79.14 ± 0.36</td>
</tr>
<tr>
<td>CCSSL [12]</td>
<td>94.86 ± 0.55</td>
<td>95.54 ± 0.20</td>
<td>75.70 ± 0.63</td>
<td>80.68 ± 0.16</td>
</tr>
<tr>
<td>SimMatch [13]</td>
<td>95.16 ± 0.39</td>
<td><strong>96.04 ± 0.01</strong></td>
<td>74.93 ± 0.32</td>
<td>79.42 ± 0.11</td>
</tr>
<tr>
<td>Ours</td>
<td><strong>95.52 ± 0.37</strong></td>
<td>95.98 ± 0.03</td>
<td><strong>75.92 ± 0.56</strong></td>
<td><strong>80.88 ± 0.32</strong></td>
</tr>
</tbody>
</table>
Algorithm 1: Pseudo-code of Feature Adversarial Training in a Pytorch-like style

```python
# f: encoder; cls: classifier
# KL: Kullback-Leibler Divergence; norm_l2: L2 Normalization
for x, _ in unselected_unlabeled_loader:
    x1, x2 = strong_aug1(x), strong_aug2(x)
    z1, z2, logit1, logit2 = f(x1), f(x2), cls(f(x1)), cls(f(x2))
    p1, p2 = torch.softmax(logit1, dim=1), torch.softmax(logit2, dim=1)

    # initialize two random noises
    d1 = torch.normal(0, 1, size = z1.shape)
    d2 = torch.normal(0, 1, size = z2.shape)
    d1, d2 = norm_l2(d1), norm_l2(d2)

    # apply random noises
    z_d1 = torch.tensor(z1.clone().detach().cpu().data + d1, requires_grad=True)
    z_d2 = torch.tensor(z2.clone().detach().cpu().data + d2, requires_grad=True)
    logit_d1, logit_d2 = cls(z_d1), cls(z_d2)
    p_d1, p_d2 = torch.softmax(logit_d1, dim=1), torch.softmax(logit_d2, dim=1)

    # generate feature-level adversarial noises
    loss_kl = KL(p_d1, p2) + KL(p_d2, p1)
    loss_kl.backward(retain_graph=True)
    adv1, adv2 = norm_l2(z_d1.grad), norm_l2(z_d2.grad)

    # calculate FAT loss
    logit_adv1, log_adv2 = cls(z1 + adv1), cls(z2 + adv2)
    p_adv1, p_adv2 = torch.softmax(logit_adv1, dim=1), torch.softmax(log_adv2, dim=1)
    loss_FAT = KL(p_adv2, p1) + KL(p_adv1, p2)
```

Algorithm 2: Pseudo-code of Virtual Adversarial Training in a Pytorch-like style

```python
# M: encoder + classifier;
for x, _ in unlabeled_loader:
    x = strong_aug(x)
    logit = M(x)
    p = torch.softmax(logit, dim=1)

    # initialize random noise
    d = torch.normal(0, 1, size = x.shape)
    d = norm_l2(d)

    # apply random noise
    x_d = torch.tensor(x + d, requires_grad=True)
    logit_d = M(x_d)
    p_d = torch.softmax(logit_d, dim=1)

    # generate image-level adversarial noise
    loss_kl = KL(p_d, p)
    loss_kl.backward(retain_graph=True)
    adv = norm_l2(x_d.grad)

    # calculate VAT loss
    logit_adv = M(x + adv)
    p_adv = torch.softmax(logit_adv, dim=1)
    loss_VAT = KL(p_adv, p)
```