Appendix

1 Model Architecture

1.1 ResNet

We utilize the standard ResNet18 and ResNet101 architectures introduced in [2] with a few modifications. The first pooling layers are removed, and the kernel size of the first convolution layers is changed to $(3, 3)$. Additionally, the linear layers are set to have an output dimension of 512.

The following PyTorch-style code demonstrates how we transform the original ResNet into the version used in our study:

```
def resnetToCifar(resnet):
    resnet.conv1 = nn.Conv2d(3, 64, (3, 3), padding='same')
    resnet.maxpool = nn.Identity()
    resnet.fc = nn.Linear(resnet.fc.in_features, 512)
    return resnet
```

1.2 Vision Transformer

The basic architecture of the Vision Transformer (ViT) we employ is consistent with the initial version presented in [1]. However, we adapt its scale to match that of ResNet101 and set the output dimension to 512. The following code defines the modified ViT:

```
def ViTCifar():
    model = VisionTransformer(image_size=32, patch_size=4,
        num_layers=16, num_heads=8, hidden_dim=512,
        mlp_dim=1536, dropout=0.2, attention_dropout=0.2)
    model.heads = nn.Linear(512, 512)
    return model
```

1.3 Graph Convolution Network

The Graph Convolutional Network (GCN) we utilize consists of three layers, including one embedding layer and two graph convolution layers proposed in [3]. The input dimension is set to 11, considering that CIFAR-10 [4] has 10 classes, and we add an additional "masked" class during training. The hidden dimension is 16, and the output dimension is 10. We also introduce a learnable scale parameter α in each convolution layer, transforming the adjacency matrix as $\hat{A} = \alpha A + I_N$.

```
class GCNhead(nn.Module):
    def __init__(self):
        self.embedding = nn.Embedding(11, 16)
        self.GCN = GCNSequential(
                   GCNWithLoop(16, 16, activation=nn.SELU()),
                   GCNWithLoop(16, 10))
```

Table 1: **Parameter Counts of the Involved Models.**

ResNet18	ResNet101	ViT	GCN	Linear Classifier
11,431,552	43,541,632	42,363,904	620	5120

```
def forward(self, A, node_feat):
    return self.GCN(A, self.embedding(node_feat))
```

The parameter counts for each model are listed in Table 1.

2 Training Recipe

We present some key hyperparameters in Table 2. It is important to note that these parameters are tuned to ensure stable training processes and maximize GPU usage, rather than achieving state-of-the-art performance.

Table 2: **Hyperparameters in the Training Recipe.**

	Contrastive			Supervised		
	ResNet18	ResNet101	ViT	ResNet18	ResNet101	ViT
Epochs	1200	1200	1200	120	120	600
Batch size	1200	700	700	1500	1000	1000
Learning rate	1.E-05	1.E-05	5.E-06	1.E-05	1.E-05	5.E-06
Optimizer	AdamW					
Learning rate scheduler	Cosine Annealing					
Warm-up scheduler	Linear					
Warm-up learning rate	1.E-03	1.E-03	1.E-03	1.E-03	1.E-03	1.E-03
Warm-up epochs	150	150	150	60	60	100
Minimum learning rate	1.E-08	1.E-08	1.E-08	1.E-08	1.E-08	1.E-08
Negative sample for contrast	128,000	128,000	128,000	N/A	N/A	N/A

References

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016.

[4] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.