Using Pre-Training Can Improve Model Robustness and Uncertainty

Reproduced by: Xingchen Liu Rongyu Wang

12/06/2019

Hendrycks, Dan & Lee, Kimin & Mazeika, Mantas. (2019). Using Pre-Training Can Improve Model Robustness and Uncertainty. CoRR. https://dblp.org/rec/bib/journals/corr/abs-1901-09960

Background

- Pre-training is widely used for deep convolutional neural networks
  - Application: “Pre-train then Tune” paradigm
  - Research: create “universal representations”

- **Doubts** on Pre-training
  - He et al. [1] argued pre-training shows no performance benefits for traditional models or large tasks.
  - Lin et al. [2] found that pre-training doesn’t have advantages when giving sufficient time for training, even for the tuning for extremely small datasets.

Motivation

- To demonstrate the effectiveness of pre-training on several aspects.

- To prove that pre-training enhances model uncertainty estimates in
  - Out-of-Distribution detection
  - calibration

Related Work

- **Pre-training**
  - Improves generalization for tasks with small datasets, including transfer learning[3][4], and tasks with significant variation[5].
  - Used in large tasks such as Microsoft COCO[1]
    - but no accuracy gains in performance[2][15]

Related Work

- **Uncertainty**
  - To detect out-of-distribution samples, use the maximum value of a classifier's softmax distribution[18]
  - Mahalanobis distance-based scores that characterize out-of-distribution samples using hidden features[9]
  - Using GAN[21] to generate out-of-distribution samples
  - Applying non-specific, real, and diverse outlier images or text instead improves out-of-distribution detection performance and calibration[22]
  - Contemporary networks can easily become miscalibrated without additional regularization[23]

Target Task

Show the performance of pre-training in:

- **Uncertainty**
  - Out-of-Distribution Detection
  - Calibration

Data Summary

- Downsampled(64 x 64) ImageNet[10] for pre-training
- CIFAR-10, CIFAR-100 and Tiny ImageNet datasets[24] without 200 overlapping Tiny ImageNet classes from Downsampled ImageNet

Implementation

- Use 40-2 Wide ResNets trained using SGD with Nesterov momentum and a cosine learning rate.

$$\eta_t = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min})(1 + \cos(\frac{T_{cur}}{T_i}\pi)),$$

- Pre-training: 100 epochs on Downsampled ImageNet, fine-tuned for 10 epochs for CIFAR and 20 for Tiny ImageNet without dropout (learning rate of 0.001).
- Baseline: 100 epochs with a dropout rate of 0.3.

Basic model

```
class WideResNet(nn.Module):
    def __init__(self, depth, num_classes, widen_factor=1, dropRate=0.0):
        super(WideResNet, self).__init__()
        nChannels = [16, 16 * widen_factor, 32 * widen_factor, 64 * widen_factor]
        assert ((depth - 4) % 6 == 0)
        n = (depth - 4) // 6
        block = BasicBlock
        # 1st conv before any network block
        self.conv1 = nn.Conv2d(3, nChannels[0], kernel_size=3, stride=1,
                               padding=1, bias=False)
        # 1st block
        self.block1 = NetworkBlock(n, nChannels[0], nChannels[1], block, 1, dropRate)
        # 2nd block
        self.block2 = NetworkBlock(n, nChannels[1], nChannels[2], block, 2, dropRate)
        # 3rd block
        self.block3 = NetworkBlock(n, nChannels[2], nChannels[3], block, 2, dropRate)
        # global average pooling and classifier
        self.bn1 = nn.BatchNorm2d(nChannels[3])
        self.relu = nn.ReLU(inplace=True)
        self.fc = nn.Linear(nChannels[3], num_classes)
        self.nChannels = nChannels[3]
```

Train from scratch

```
# Train network on CIFAR10
train_data, test_data = load_cifar10_data()
model = WideResNet(40, 10, 2, 0.3)
model.cuda()
total_steps = 100 * len(train_data)
optimizer = torch.optim.SGD(model.parameters(), 0.1, momentum=0.9, nesterov=True)
scheduler = torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda= lambda step: cosine_lr(step, total_steps, 1, 10e-5))
save_name = "CIFAR10_baseline.pt"
train_process(model,train_data, test_data, optimizer, scheduler, 100,save_name)
```

Pre-train then tune

```
# Pre-train on ImageNet Downsample
torch.cuda.manual_seed(1)
train_data, test_data = load_imagenetds_data()
model = WideResNet(40, 10, 2, 0.3)
model.cuda()
total_steps = 100 * len(train_data)
optimizer = torch.optim.SGD(model.parameters(), 0.1, momentum=0.9, nesterov=True)
scheduler = torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda= lambda step: cosine_lr(step, total_steps, 1, 10e-5))
save_name = "Tiny_ImageNet_baseline.pt"
train_process(model,train_data, test_data, optimizer, scheduler, 100, save_name)
```

```
# Tune on CIFAR10
train_data, test_data = load_cifar10_data()
model = WideResNet(40, 1000, 2, 0)
model.cuda()
model.load_state_dict(torch.load("saved_model/pretrained_model.pt"))
total_steps = 100 * len(train_data)
optimizer = torch.optim.SGD(model.parameters(), 0.001, momentum=0.9, nesterov=True)
scheduler = torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda= lambda step: cosine_lr(step, total_steps, 1, 10e-5))
save_name = "CIFAR10_pretrained.pt"
train_process(model,train_data, test_data, optimizer, scheduler, 10, save_name)
```

Out-of-Distribution Detection

- In distribution data: test dataset
- Out-of-distribution data: Gaussian noise, textures etc.
- Use the maximum softmax probabilities to score anomalies
- Evaluation metrics: AUROC (the Area Under the Receiver Operating Characteristic curve), AUPR (the Area Under the Precision-Recall Curve)

```python
def get_measures(out_score, in_score):
    out_score = np.array(out_score)
    in_score = np.array(in_score)
    examples = np.concatenate((out_score, in_score), axis=None)
    labels = np.zeros(len(examples), dtype=np.int32)
    labels[:len(out_score)] += 1

    auroc = sk.roc_auc_score(labels, examples)
    aupr = sk.average_precision_score(labels, examples)
    return auroc, aupr

def get_results(ood_loader,auroc_results, aupr_results):
    aurocs, auprs = [], []
    for i in range(5):
        out_score = get_scores(ood_loader)
        auroc, aupr = get_measures(out_score, in_score)
        aurocs.append(auroc)
        auprs.append(aupr)

    auroc = np.mean(aurocs)
    aupr = np.mean(auprs)
    auroc_results.append(auroc*100)
    aupr_results.append(aupr*100)
    print("AUROC:",100*auroc)
    print("AUPR:",100*aupr)
    print("\n\n")
```

Calculate AUROC and AUPR

Results in Paper

Out-of-distribution Detection

	AUROC		AUPR	
	Normal	Pre-Train	Normal	Pre-Train
CIFAR-10	91.5	94.5	63.4	73.5
CIFAR-100	69.4	83.1	29.7	52.7
Tiny ImageNet	71.8	73.9	30.8	31.0

Reproduction Results

Out-of-distribution Detection for CIFAR10

CIFAR10	AUROC		AUPR	
	Normal	Pre-training	Normal	Pre-training
Gaussian	89.32	**95.44**	43.44	**64.38**
Rademacher	89.04	**94.61**	42.79	**59.47**
Blob	94.56	**97.15**	68.11	**83.67**
Textures	87.98	**93.74**	55.84	**70.85**
SVHN	91.77	**95.65**	63.47	**76.77**
CIFAR100	87.42	**90.49**	54.76	**65.30**
Mean	90.02	**94.49**	54.80	**69.94**

Reproduction Results

Out-of-distribution Detection for CIFAR100

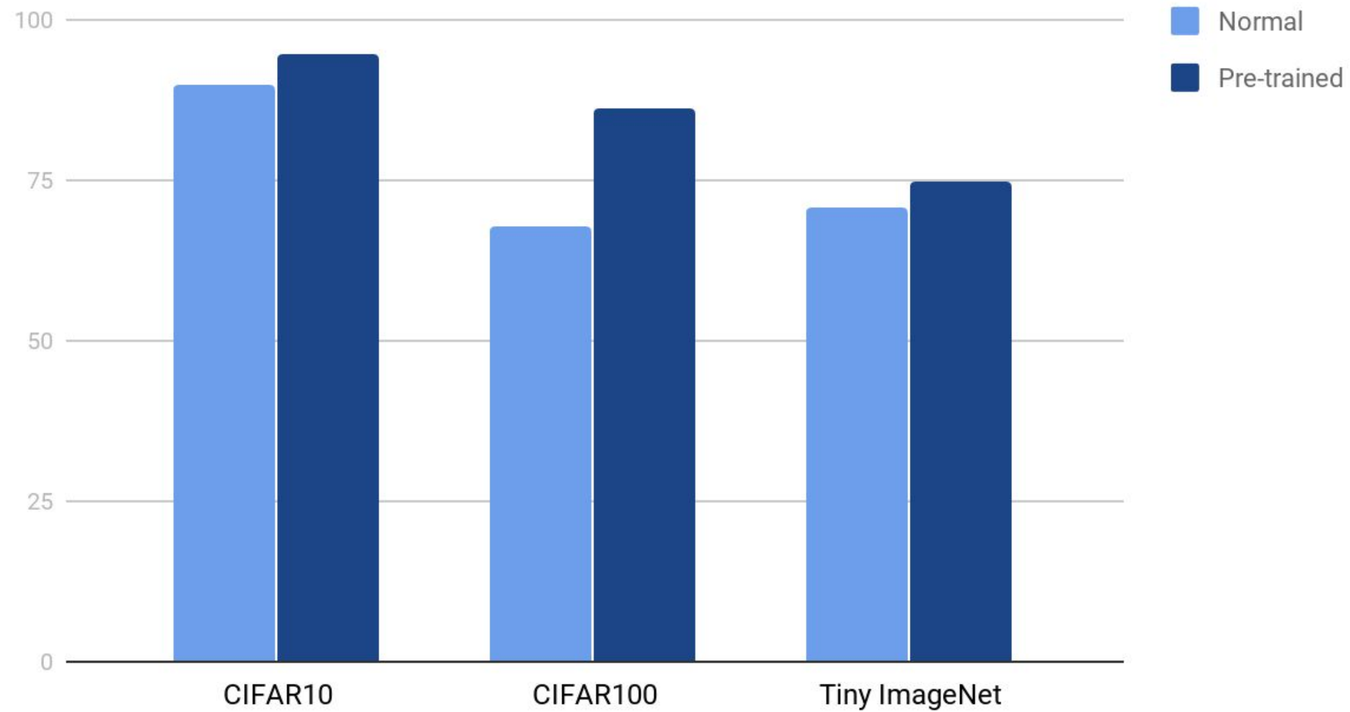
CIFAR100	AUROC		AUPR	
	Normal	Pre-training	Normal	Pre-training
Gaussian	49.67	**96.61**	14.78	**79.95**
Rademacher	46.75	**97.66**	14.13	**87.54**
Blob	85.89	**89.34**	45.02	**55.96**
Textures	73.29	**79.65**	33.03	**44.03**
SVHN	74.51	**79.21**	32.27	**48.42**
CIFAR10	75.66	**75.21**	34.71	**35.52**
Mean	67.63	**86.28**	28.99	**58.57**

Reproduction Results

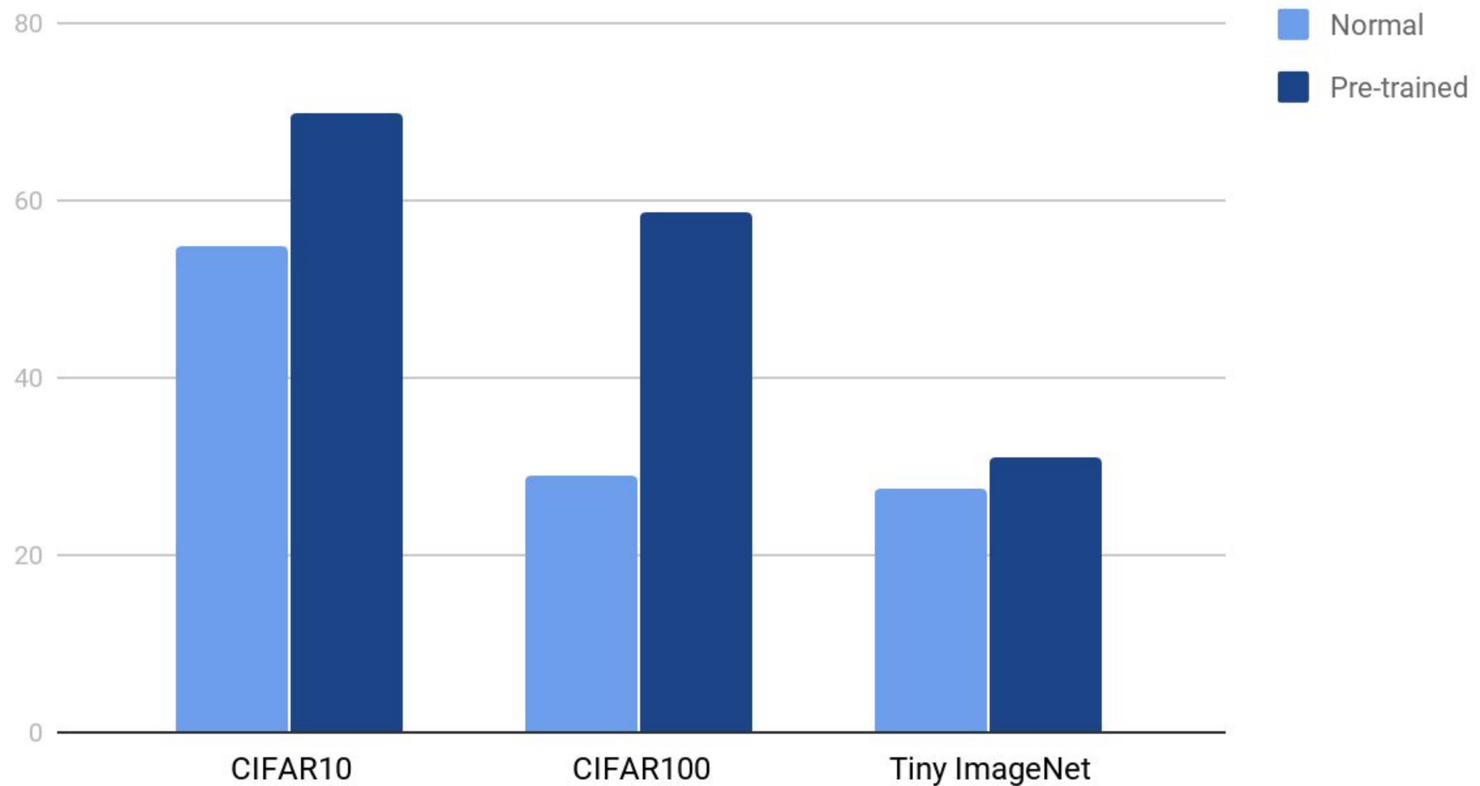
Out-of-distribution Detection for Tiny ImageNet

Tiny ImageNet	AUROC		AUPR	
	Normal	Pre-training	Normal	Pre-training
Gaussian	64.82	**78.01**	18.68	**26.65**
Rademacher	67.48	**73.42**	19.86	**23.17**
Blob	**64.84**	60.47	**18.95**	17.14
Textures	68.99	**72.25**	29.02	**29.99**
SVHN	86.66	**89.24**	51.39	**57.69**
Mean	70.58	**74.68**	27.58	**30.93**

Reproduction Results



Reproduction Results



Calibration

- According to methods of Chawla *et al*[22], adopt RMS and MAD to measure the calibration of a classifier

RMS: $$\sqrt{\sum_{i=1}^{b} \frac{|B_i|}{n} \left(\frac{1}{|B_i|} \sum_{k \in B_i} \mathbb{1}(y_k = \widehat{y}_k) - \frac{1}{|B_i|} \sum_{k \in B_i} c_k \right)^2}.$$

MAD: $$\sum_{i=1}^{b} \frac{|B_i|}{n} \left| \frac{1}{|B_i|} \sum_{k \in B_i} \mathbb{1}(y_k = \widehat{y}_k) - \frac{1}{|B_i|} \sum_{k \in B_i} c_k \right|.$$

Soft F1 score: $$\frac{c_a^{\mathsf{T}} m}{1^{\mathsf{T}}(c_a + m)/2}$$

- Compare results of using pre-training and not using pre-training.

Results in paper

In-distribution Calibration

	RMS Error		MAD Error	
	Normal	Pre-Train	Normal	Pre-Train
CIFAR-10	6.4	2.9	2.9	1.2
CIFAR-100	13.3	3.6	10.3	2.5

Out-of-distribution data

```python
# SVHN
svhn_transform = trn.Compose([trn.Resize(dim), trn.ToTensor(), trn.Normalize(mean,std)])
svhn_data = SVHN(root="data/SVHN", download = True, split="test", transform=svhn_transform)
svhn_loader = torch.utils.data.DataLoader(svhn_data, batch_size=200, shuffle=True)
print('SVHN Calibration')
get_results(svhn_loader,auroc_results, aupr_results)

# Texture
texture_transform = trn.Compose([trn.Resize(dim), trn.CenterCrop(dim),trn.ToTensor(), trn.Normalize(mea
texture_data = dset.ImageFolder(root="data/dtd/images",transform=texture_transform)
texture_loader = torch.utils.data.DataLoader(texture_data, batch_size=200, shuffle=True,num_workers=4,

print('Texture Detection')
get_results(texture_loader,auroc_results, aupr_results)
```

```python
# CIFAR
if name == "CIFAR10":
    cifar_data = dset.CIFAR100('data/CIFAR100', train=False, transform=test_transform)
    cifar_loader = torch.utils.data.DataLoader(cifar_data, batch_size=200, shuffle=True)
    print('CIFAR Detection')
    get_results(cifar_loader,auroc_results, aupr_results)

if name == "CIFAR100":
    cifar_data = dset.CIFAR10('data/CIFAR10', train=False, transform=test_transform)
    cifar_loader = torch.utils.data.DataLoader(cifar_data, batch_size=200, shuffle=True)
    print('CIFAR Detection')
    get_results(cifar_loader,auroc_results, aupr_results)

# Mean Result
print(auroc_results)
auroc_mean = sum(auroc_results) / len(auroc_results)
aupr_mean = sum(aupr_results) / len(aupr_results)
print('Mean Results')
```

```
# Gaussian Noise
gaussian_targets = torch.ones(ood_num * 5)

m = normal.Normal(0.5, 0.5)
gaussian_ood = m.sample((ood_num * 5,3,dim,dim))

gaussian_ood = torch.utils.data.TensorDataset(gaussian_ood, gaussian_targets)
gaussian_loader = torch.utils.data.DataLoader(gaussian_ood, batch_size=200,shuffle=True)

print('Gaussian Noise Detection')
get_results(gaussian_loader,auroc_results, aupr_results)

# Rademacher Noise

rademacher_targets = torch.ones(ood_num * 5)
m = bernoulli.Bernoulli(0.5)
rademacher_ood  = m.sample((ood_num * 5, 3, dim, dim))
rademacher_data = torch.utils.data.TensorDataset(rademacher_ood, rademacher_targets)
rademacher_loader = torch.utils.data.DataLoader(rademacher_data, batch_size=200, shuffle=True)

print('Rademacher Noise Detection')
get_results(rademacher_loader,auroc_results, aupr_results)
```

Out-of-distribution data

```python
def rms_mad_error(confidence, correct, error_name, bin_size):
  index = np.argsort(confidence)
  confidence = confidence[index]
  correct = correct[index]
  num_bins = len(confidence) // bin_size
  bins = [i * bin_size for i in range(num_bins)]

  error_count = 0
  num_samples = len(confidence)
  for i in range(num_bins):
    if i<num_bins-1:
      start,end = bins[i],bins[i+1]
    if i == num_bins-1:
      start,end = bins[i],len(confidence)
    bin_conf = confidence[start:end]
    bin_corr = correct[start:end]
    bin_size = len(bin_conf)

    if bin_size > 0:
      diff = np.abs(np.nanmean(bin_conf) - np.nanmean(bin_corr))
      if error_name == 'rms':
        error_count += bin_size / num_samples * np.square(diff)
      elif error_name == 'mad':
        error_count += bin_size / num_samples * diff

  if error_name == 'rms':
    error_count = np.sqrt(error_count)

  return error_count
```

Calculate RMS, MAD, Sf1

```python
def SoftF(confidence, correct):
    recall = 1 - correct
    precision = 1 - confidence
    num = (precision * recall).sum()
    denom = (precision + recall).sum()/2
    sf1 = num / denom
    return sf1
```

Calculate RMS, MAD, Sf1

Reproduction Results

Calibration for CIFAR10

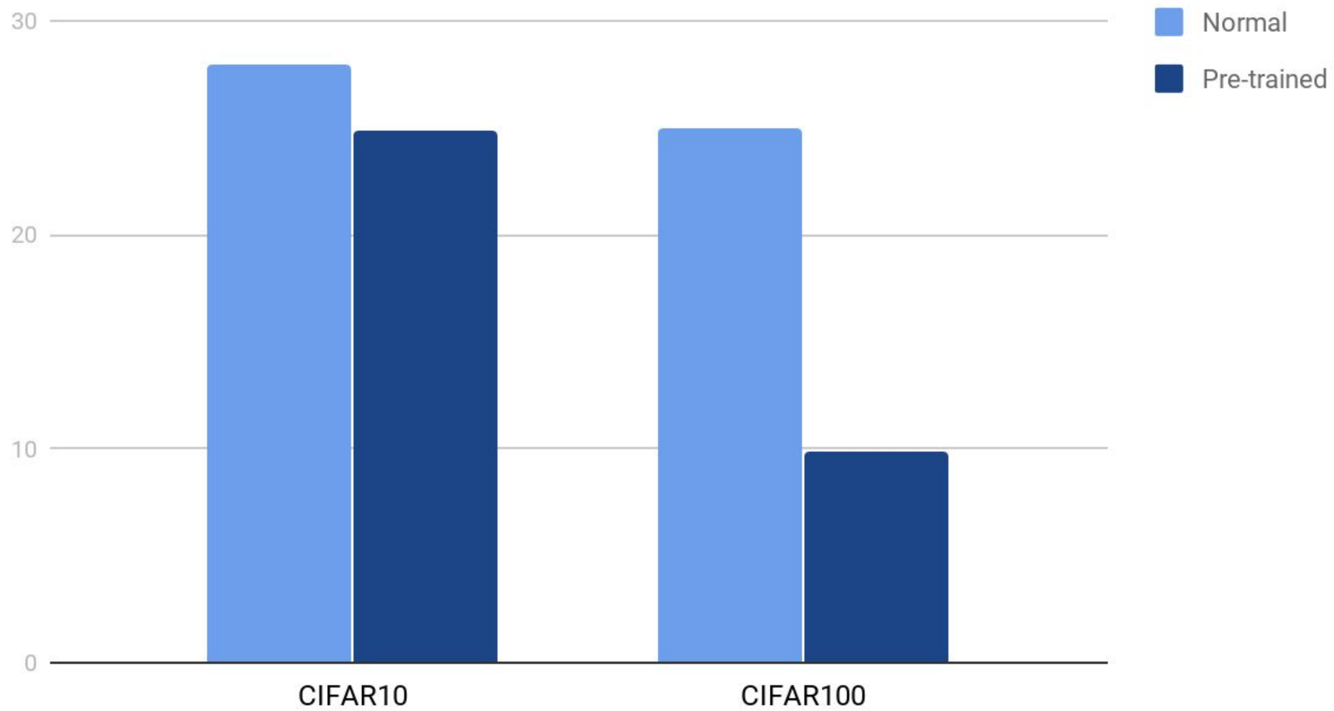
CIFAR10	RMS Error		MAD Error		Soft F1 Score	
	Normal	Pre-training	Normal	Pre-training	Normal	Pre-training
In-Distribution	6.39	**2.88**	2.87	**1.24**	27.83	**29.65**
Gaussian	33.02	**28.17**	18.09	**14.06**	15.44	**34.63**
Rademacher	33.29	**29.63**	18.20	**14.73**	14.61	**30.12**
Blob	26.76	**22.54**	14.94	**11.73**	36.69	**48.46**
Textures	24.83	**23.18**	16.02	**13.46**	29.88	**38.08**
SVHN	25.26	**23.30**	15.37	**12.74**	34.01	**42.58**
CIFAR100	24.80	**22.14**	16.26	**13.82**	28.31	**35.69**
Mean	24.91	**21.69**	14.54	**11.69**	26.68	**37.03**

Reproduction Results

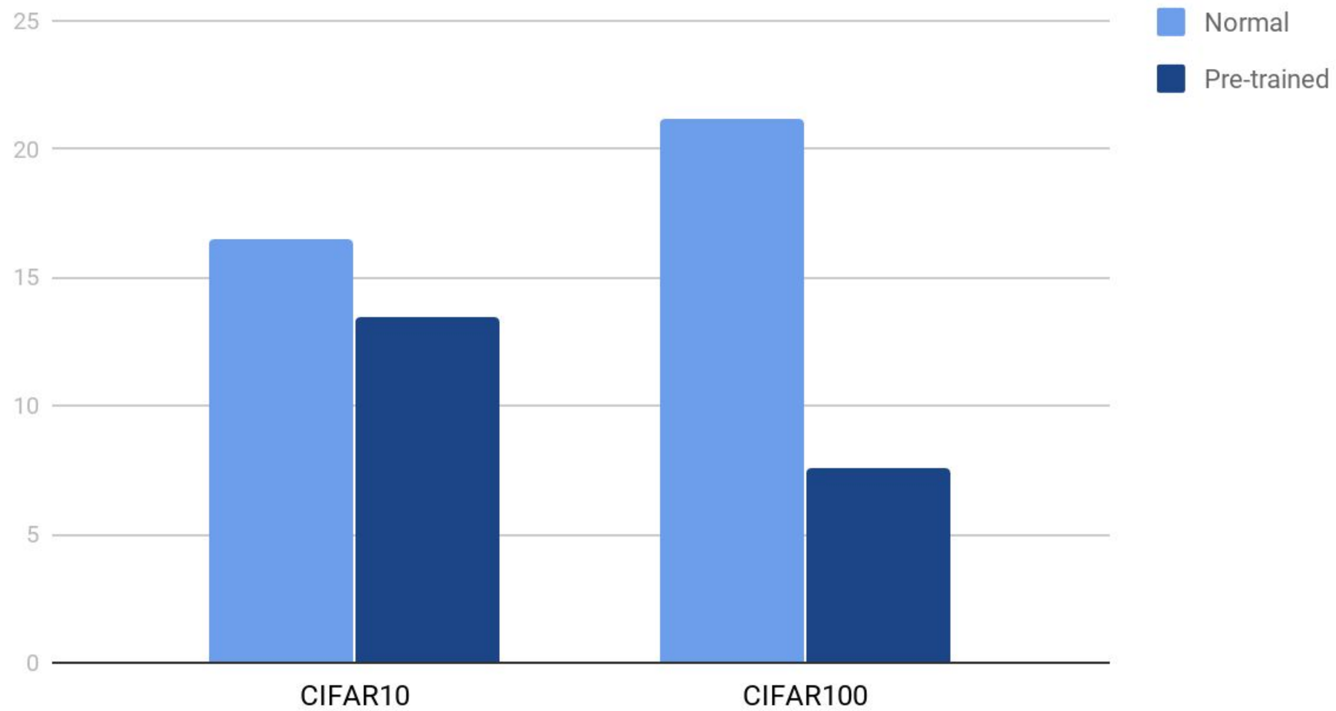
Calibration for CIFAR100

CIFAR100	RMS Error		MAD Error		Soft F1 Score	
	Normal	Pre-training	Normal	Pre-training	Normal	Pre-training
In-Distribution	13.32	**3.63**	10.26	**2.51**	42.44	**46.33**
Gaussian	28.25	**8.48**	24.30	**5.70**	30.90	**64.29**
Rademacher	28.11	**7.49**	24.52	**5.10**	30.16	**65.71**
Blob	22.62	**10.13**	17.95	**7.31**	50.35	**59.27**
Textures	23.40	**10.81**	20.11	**8.76**	44.24	**54.32**
SVHN	24.06	**10.23**	24.06	**8.60**	44.04	**54.67**
CIFAR10	23.64	**11.48**	19.91	**9.60**	44.85	**52.12**
Mean	23.34	**8.89**	19.61	**6.80**	41.00	**56.67**

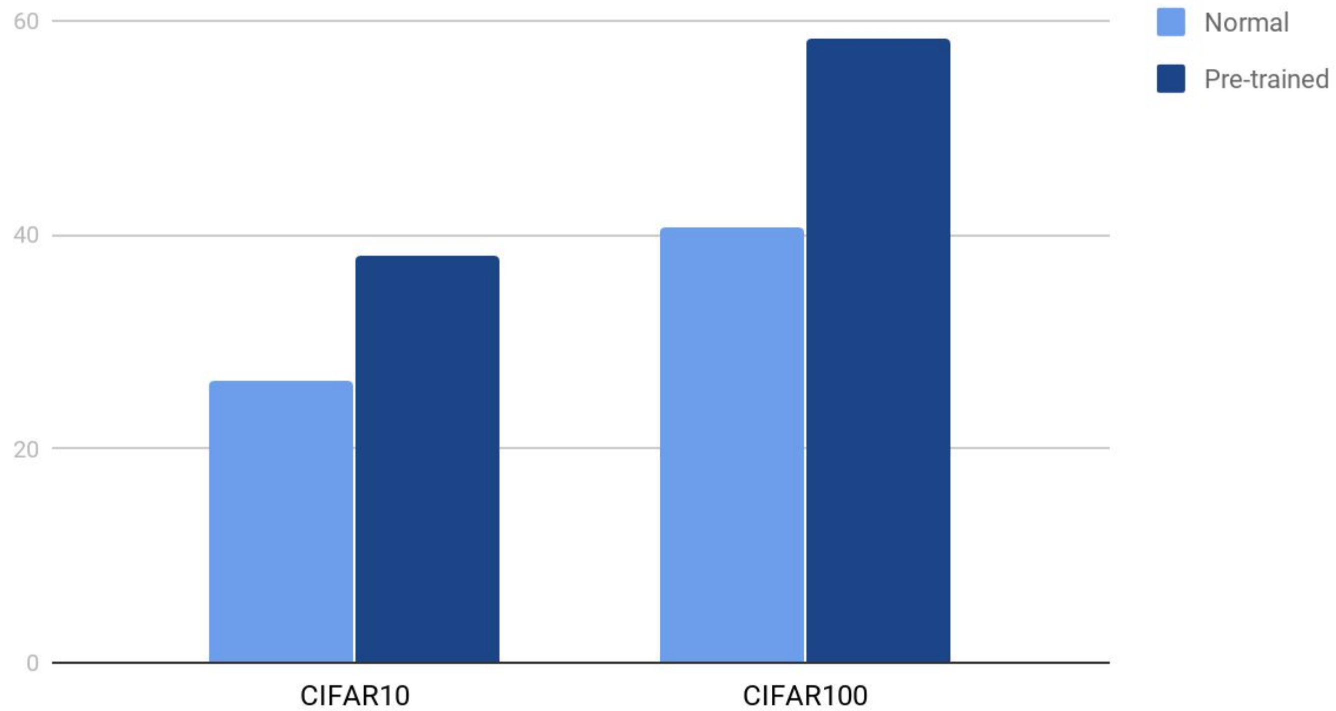
Reproduction Results



Reproduction Results



Reproduction Results



Conclusion and Future Work

Conclusion

- The benefits of pre-training extend beyond merely quick convergence, as previously thought, since pre-training can improve model uncertainty.
  - Pre-trained representations directly translate to improvements in predictive uncertainty estimates.
  - Training from scratch can only reach the same performance as training with pre-training on unperturbed data.

Conclusion and Future Work

Future Work

- Figure out the reasons for unexpected lowerer performance of pre-trained models
  - OOD detection(Tiny-ImageNet) on Blob
- Reproduce the part of evaluation on robustness of models.
- Validate the effects of pre-training on other datasets.
- Compare the effects of different strategy for pre-training
- Some work could specialize pre-training for these downstream tasks.

Job Split

Xingchen Liu:

Train baseline network from scratch, OOD detection

Clare Wang:

Pre-train network and tune, Calibration

References

[1] He, K., Girshick, R., and Dollar, P. Rethinking ImageNet pre-training. arXiv, 2018.
[2] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollar, P. Microsoft COCO: Common objects in context. ECCV, 2014.
[3] Agrawal, P., Girshick, R., and Malik, J. Analyzing the performance of multilayer neural networks for object recognition. ECCV, 2014.
[4] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? NeurIPS, 2014.
[5] Huh, M., Agrawal, P., and Efros, A. A. What makes ImageNet good for transfer learning? arXiv, 2016.
[6] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. ICLR, 2018.
[7] Zhang, Z. and Sabuncu, M. Generalized cross entropy loss for training deep neural networks with noisy labels. NeurIPS, 2018.
[8] Hendrycks, D. and Gimpel, K. A baseline for detecting misclassified and out-of-distribution examples in neural networks. ICLR, 2017b.
[9] Lee, K., Lee, K., Lee, H., and Shin, J. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. NeurIPS, 2018b.

References

[10] Chrabaszcz, P., Loshchilov, I., and Hutter, F. A downsampled variant of ImageNet as an alternative to the CIFAR datasets. arXiv, 2017.
[11] Dong, Q., Gong, S., and Zhu, X. Imbalanced deep learning by minority class incremental rectification. IEEE TPAMI, 2018.
[12] Japkowicz, N. The class imbalance problem: Significance and strategies. In ICAI, 2000.
[13] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. SMOTE: synthetic minority over- sampling technique. JAIR, 2002.
[14] Huang, C., Li, Y., Change Loy, C., and Tang, X. Learning deep representation for imbalanced classification. In CVPR, 2016.
[15] Sun, C., Shrivastava, A., Singh, S., and Gupta, A. Revisiting unreasonable effectiveness of data in deep learning era. ICCV, 2017.
[16] Sukhbaatar, S., Bruna, J., Paluri, M., Bourdev, L., and Fergus, R. Training convolutional networks with noisy labels. ICLR Workshop, 2014.
[17] Patrini, G., Rozza, A., Menon, A., Nock, R., and Qu, L. Making deep neural networks robust to label noise: a loss correction approach. CVPR, 2017.
[18] Hendrycks, D., Mazeika, M., Wilson, D., and Gimpel, K. Using trusted data to train deep networks on labels corrupted by severe noise. NeurIPS, 2018.

References

[19]He, H. and Garcia, E. A. Learning from imbalanced data. TKDE, 2008.
[20] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. SMOTE: synthetic minority over- sampling technique. JAIR, 2002.
[21] Lee, K., Lee, H., Lee, K., and Shin, J. Training confidence- calibrated classifiers for detecting out-of-distribution samples. ICLR, 2018a.
[22] Hendrycks, D., Mazeika, M., and Dietterich, T. Deep anomaly detection with outlier exposure. ICLR, 2019.
[23] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. International Conference on Machine Learning, 2017.
[24] Johnson et al. Tiny ImageNet visual recognition challenge. URL https://tiny-imagenet. herokuapp.com.