Project List

1) Examining Lottery Ticket Hypothesis (Lead TA: Huanrui Yang)
2) Input-dependent Dynamic CNN model (Lead TA: Huanrui Yang)
3) TernGrad and Gradient Manipulation During Training (Lead TA: Huanrui Yang)
4) Noise aware DNN training (Lead TA: Huanrui Yang)
5) Robust and Non-robust Features (Lead TA: Matt Inkawhich)
6) Hidden Feature-Based Adversarial Attacks (Lead TA: Matt Inkawhich)
7) Improved Regularization of Convolutional Neural Networks (Lead TA: Matt Inkawhich)
8) Out-of-Distribution Detection (Lead TA: Jingyang Zhang)
9) Self-Supervised / Representation Learning (Lead TA: Jingyang Zhang)
10) Adversarial Patch Attack (Lead TA: Jingyang Zhang)
11) Mage Synthesis with A Single (Robust) Classifier (Lead TA: Jingyang Zhang)
12) Neural Netowrk Distillation (Lead TA: Meng Xia)
13) Dynamic BlockDrop (Lead TA: Meng Xia)
14) Conditional Generative Adversarial Networks (Lead TA: Meng Xia)
15) Self-supervised Learning for Language Modeling (Lead TA: Yuan Li)
16) Learned Non-linear Quantization (Lead TA: Congjia Yu)
17) Compressing the Transformer Model (Lead TA: Congjia Yu)
18) Data poison attack to Federated Learning (Lead TA: Hongyu He)
19) DeepFake Generation (Lead TA: Hongyu He)
20) Decision-based Black-box Adversarial Attacks (Lead TA: Hongyu He)
21) ***You own proposed project (talk to TA’s to finalize the details)***

Project 1: Examining Lottery Ticket Hypothesis

Lead TA: Huanrui Yang

Project Description:

It is commonly believed that a larger DNN model (with more parameters or more layers) can reach a higher accuracy comparing to a smaller model under the same training scheme, thanks to the larger capacity of the model. This is the reason why people tend to prune a large DNN model rather than directly training a smaller one from scratch. However, with a specially designed initialization scheme and model architecture, a small model can possibly reach a comparable accuracy as the larger ones. Formally, this is stated as the “Lottery Ticket Hypothesis”: Dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation—reach test accuracy comparable to the original network in a similar number of iterations. The hypothesis as well as methods for finding the winning tickets are proposed in [1], which earn it the best paper award of ICLR 2019. You will be asked to read the paper, follow the proposed method to find the winning tickets, and examine the correctness of the Lottery Ticket Hypothesis.

Basic Requirements

- With VGG-16 or ResNet-20 model on CIFAR-10 dataset, implement the proposed algorithm for identifying winning ticket initializations, test the training performance under different sparsity
- Compare the achieved accuracy of the winning ticket with the original model, original model pruned (with iterative pruning) to the same sparsity, and randomly reinitialize the winning ticket
- Compare the difference between finding winning ticket with layer-by-layer pruning vs. global pruning, as discussed in Section 4
- Try different masking criterion other than the absolute value proposed in [1] (See [2] (NeurIPS 2019) for some ideas, you can also try saliency-based criterion introduced in the lecture or propose your own)
- Compare the results and find the best way for finding the winning tickets

Optional Requirements (Choose the one you are interested):

- Examine if winning ticket initialization can be found for PGD adversarial training

OR

- Finding lottery ticket with Continuous Sparsification [3]

References:

[1] Frankle et al., “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks,” (https://openreview.net/pdf?id=rJl-b3RcF7)

[2] Zhou et al., “Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask,” (https://arxiv.org/pdf/1905.01067.pdf)

[3] Savarese et al., “Winning the Lottery with Continuous Sparsification” (https://arxiv.org/pdf/1912.04427.pdf)

Project 2: Input-dependent Dynamic CNN model

Lead TA: Huanrui Yang

Project Description:

All the model compression algorithms we discuss in this course consider inducing a smaller model architecture to be used replacing the original model in the entire learning task. However, different aspects of a learning task (i.e. identifying different classes) may require totally different features. As all these features will present in the final learned model, there's no need to call all of them to deal with each individual input. This leads us to the idea of input-dependent dynamic model. In this project we will examine two lines of works in this field: dynamic kernel combination [1] and dynamic filter selection [2].

Basic Requirement:

- Dynamic convolution proposed in [1] allows having more trainable parameters (convolutional kernels) in the model without significantly increase the inference-time FLOPs. Read [1], and implement a ResNet-20 model with all the convolutional layers replaced by the dynamic convolution layer proposed.
- Train the model on CIFAR-10 dataset, start with the original ResNet-20 model, then train with 2, 3, 4 and 5 kernels in each layer respectively. Record how training and validation accuracy change as the number of kernels increases.
- Select some layers to observe the kernel attention map distribution given different inputs. Record the attention map achieved with each testing image and observe it then can be distinguished based on the class of the testing image. You may use visualization tools like T-SNE to help your observation.
- Read and implement [2] on ResNet-20, where only part of the channels is used in each layer at test time, reducing FLOPs without shrinking the model
- Report the accuracy and averaged FLOPs on CIFAR-10 validation set, visualize the gate distribution in a similar way of Figure 5 and Figure 6 in the paper

Optional Requirements (Choose the one you are interested):

- Apply white-box PGD adversarial attack to the input of [1] or [2], how is the attention map/gate selection change comparing to that of the clean input? What's the case for a transferred black-box attack?

OR

- Perform PGD adversarial training on [1] or [2], compare performance with naïve models?

References:

[1] Chen et al., "Dynamic Convolution: Attention over Convolution Kernels," (CVPR 2020) (https://arxiv.org/pdf/1912.03458.pdf)

[2] Bejnordi et al., "Batch-Shaping For Learning Conditional Channel Gated Networks," (ICLR 2020) (https://arxiv.org/pdf/1907.06627.pdf)

Project 3: TernGrad and Gradient Manipulation During Training

Lead TA: Huanrui Yang

Project Description:

In a normal training process, we update all the weights of the model after every step of gradient computation. However, this may not be possible for all the scenarios. For example, training a model in distributed setting may be bottlenecked by the gradient communication between different machines. Some computation hardware may have specific memory device which cannot be written for multiple times during the training process. To tackle these issues, TernGrad [1] shows that convergence is possible even if the full gradient is not always presented. We will explore this method in this project

Basic Requirement:

- Build baseline for a data parallel distributed training
- Train on CIFAR-10 dataset, start with the original ResNet-20 model or a model of your choice, record baseline performance.
- Apply TernGrad to the training process, record performance.
- In some case, we may want the gradient update to be sparse. Use principle of TernGrad, propose a training algorithm that can control the number of gradient updates on each weight element. (Gradient can be full precision for this task)
- Design a constraint on the number of gradient updates for each weight element, train the model with your method, compare the total number of gradient updates with baseline method.

Optional Requirements (Choose the one you are interested):

- Observing the pattern of gradient update frequency among different layer/filter, see if this can be used as an early stopping criterion for the update of part of the model?

OR

- Find mixed-precision policy to apply different precision/sparsity to different layer's gradient for better accuracy/efficiency tradeoff.

OR

- Explore the limit of TernGrad, by checking how the final accuracy change under different gradient precision or gradient sparsity.

References:

[1] Wen et al., "TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning," (https://arxiv.org/pdf/1705.07878.pdf)

Project 4: Noise-aware DNN Training

Lead TA: Huanrui Yang

Project Description:

In our course, we explored the perturbation, especially adversarial perturbation, on the input to the neural network. Meanwhile, the perturbation on the model weight is not well explored. The performance under weight perturbation actually reflects some important properties of the model. For example, the performance under L2 bounded perturbation reflects the generalizability of the model, while the performance under L infinity bounded perturbation reflects how robust the model is against quantization. This project explores sharpness aware minimization [1], which is an effective way of improving model's robustness against weight perturbation.

Basic Requirement:

- Train a baseline model on CIFAR-10 dataset, evaluate its performance evaluation under weight perturbation of Gaussian noise or adversarial noise
- Start with naïve noise aware training, add Gaussian noise to the weight during training.
- Perform weight training with adversarial noise as in SAM.
- Compare performance on generalizability, quantization, and resistance against Gaussian weight perturbation of naïve SGD vs. Gaussian vs. SAM

Optional Requirements (Choose the one you are interested):

- Try SAM with multi-step weight perturbation, how is it affecting training performance?

OR

- Try SAM with L infinity bounded perturbation. Is it more helpful on quantization?

References:

[1] Foret et al., "Sharpness-Aware Minimization for Efficiently Improving Generalization," (https://arxiv.org/pdf/2010.01412.pdf)

Project 5: Robust and Non-Robust Features

Lead TA: Matthew Inkawhich

Project description:

It has been recently shown that both robust and non-robust features exist in common image datasets for DNN training [1]. DNN models tend to capture only the non-robust features, which are highly correlated with the class label but are highly sensitive to small noises. Such dependency on non-robust features leads to the lack of robustness against adversarial attacks and leads to the easy transfer of attacks between models trained on the same dataset. In this project, we will use the methods proposed in [1] to extract the robust and non-robust features of CIFAR-10 and observe their properties.

Requirements:

- Train a ResNet-20 model on the CIFAR-10 dataset without adversarial training. Use the algorithm described in Appendix C.5 to achieve the non-robust dataset for the CIFAR-10 training set.
- Visually observe the data under each class in the non-robust dataset. Which class do they look like? Which class does the original ResNet-20 model classify them into?
- Train a new ResNet-20 model with the non-robust dataset you got, test its validation accuracy and robustness on the original validation set.
- Train a robust ResNet-20 model on the CIFAR-10 dataset with PGD adversarial training.
- Acquire the robust feature set of the CIFAR-10 training set using your adversarially-trained model following Section 3.1 (and Appendix C.4). What do the images in the robust dataset look like?
  - Try starting from (1) a randomly sampled different image, and (2) random noise.
- Train a new ResNet-20 model with the robust dataset you got, test its validation accuracy and robustness on the original validation set.

References:

[1] Ilyas et al., "Adversarial Examples Are Not Bugs, They Are Features," (NeurIPS 2019) (https://arxiv.org/pdf/1905.02175.pdf)

Project 6: Hidden-Feature Attacks and Defenses

Lead TA: Matthew Inkawhich

Project description:

As most adversarial attacks use the final output or loss of the DNN as the objective to mislead the model, the hidden features (i.e., outputs of the intermediate layers) may convey more information than the final output. Recent research shows that generating targeted attack using the hidden feature information can lead to a higher black-box transfer rate compared to other attack methods [1]. Using such a transfer attack in a round-robin adversarial training can also produce DNN ensembles with higher black-box robustness [2]. We will explore these methods in this project.

Requirements:

- Implement the feature space attack following Section 4.2 of [1].
- Generate the feature space attack against one ResNet-20 model trained on the CIFAR-10 dataset, then transfer it to attack another ResNet-20 model. Report the error rate, untargeted transfer rate and targeted transfer rate (as defined in Section 3) vs. the strength of the attack, like the results in Figure 2. Compare the result with a PGD attack of the same strength.
- Change the layer on which the attack is generated, report how the performance changes as we change the "attack" layer, like the results in Figure 3.
- Starting with 3 normally trained ResNet-20 models, implement the round-robin adversarial training proposed in Algorithm 1 of [2]. Here the non-robust feature distillation is very similar to the feature space attack in [1].
- Evaluate the transfer attack success rate between each pair of the models, as shown in Figure 3 of [2].
- Use the average logits of the 3 models as the ensemble output, evaluate the ensemble robustness under transfer attacks with different strengths, as shown in Figure 4 of [2].

References:

[1] Inkawhich et al., "Feature Space Perturbations Yield More Transferable Adversarial Examples," (https://openaccess.thecvf.com/content_CVPR_2019/papers/Inkawhich_Feature_Space_Perturbations_Yield_More_Transferable_Adversarial_Examples_CVPR_2019_paper.pdf)

[2] Yang et al., "DVERGE: Diversifying Vulnerabilities for Enhanced Robust Generation of Ensembles", (https://arxiv.org/pdf/2009.14720.pdf)

Project 7: Improved Regularization of Convolutional Neural Networks

Lead TA: Matthew Inkawhich

Project description:

Convolutional neural networks (CNNs) are capable of learning powerful feature spaces. However, due to the mismatch in task complexity and model capacity, CNNs are prone to overfitting and thus require proper regularization to generalize well. When trained with standard cross entropy objectives, CNNs often struggle when the test distribution does not exactly align with the training distribution. Also, the models are often poorly calibrated, meaning that the confidence predictions they produce do not align with the true uncertainty. In this project, you will investigate and implement three techniques for improving model robustness to input noise and corruptions. You will implement the cutout regularizer [1], the mixup regularizer [2], and the self-supervised training method in [3]. After finding good configurations for these regularizers, you will compare the performance of these approaches for handling naturally occurring image noise and other corruptions.

Requirements (for an individual student):

- Train ResNet-20 [4] with cross-entropy loss (as usual) on CIFAR-10.
- Re-train the ResNet with the described cutout regularizer [1] on CIFAR-10. What are the key hyperparameters? How do these affect test accuracy?
- Re-train the ResNet with the described mixup regularizer [2] on CIFAR-10. What are the key hyperparameters? How do these affect test accuracy?
- Implement the auxiliary rotation head from [3] on a ResNet-20 model. Then train this model using the rotation head for self-supervision on CIFAR-10. What are the key hyperparameters? How much do these affect the test accuracy?
- How do these methods compare in terms of test-set accuracy? Is there a benefit to combining these techniques? What is the maximum test accuracy you can achieve?
- Test these models on a corrupted variant of the CIFAR-10 test set. Choose 5 (already implemented) corruptions from [5] to apply to each image in the test set and observe the accuracy of all models. Which performs the best? Which performs the worst?
- Finally, test robustness of the best regularized models to white-box FGSM [6] adversarial examples with various epsilon values. Do any of these techniques provide robustness to attack?

References:

[1] DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*.

[2] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, David Lopez-Paz: mixup: Beyond Empirical Risk Minimization. ICLR (Poster) 2018

[3] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, Dawn Song: Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty. NeurIPS 2019

[4] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[5] https://github.com/bethgelab/imagecorruptions

[6] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Project 8: Out-of-Distribution Detection

Lead TA: Jingyang Zhang

Project description:

Most of the DNN classifiers we have discussed in this class are trained to be maximally accurate on the training dataset. However, is it safe to assume that during deployment the DNN will only encounter samples from the training class set? Further, what happens if a trained CIFAR-10 classifier encounters an MNIST digit? Most current networks will output a confident prediction, even though it has no real meaning. This is a realistic concern in ML operational settings and warrants research on how to distinguish "in-distribution" and "out-of-distribution" (OOD) samples at test time. In this project, you will implement several OOD detection algorithms and evaluate their performance against a variety of OOD datasets.

Requirement:

- Implement Baseline [1], ODIN [2], and Energy [3] methods. Your ODIN method do *not* need to include the input-preprocessing step.
- Reproduce the results in the ODIN paper using CIFAR-10 as the ID dataset and CIFAR-100, SVHN, LSUN, MNIST, Uniform-Random, Gaussian-Random as the OOD datasets. Use the AUROC and TNR@95TPR metrics. Are these OOD datasets difficult to be detected? (https://github.com/pokaxpoka/deep_Mahalanobis_detector/blob/master/calculate_log.py)
- Try using models with different capacities (e.g., changing ResNet's depth and width). Does the base model capacity affect results?
- There are also dedicated training methods to enable better detection. Implement Outlier Exposure [4]. Here, only use the first five classes of CIFAR-10 (denote as CIFAR-5) as ID classes, and use CIFAR-100 as the outlier dataset. After training, use the second five classes of CIFAR-10, SVHN, LSUN, MNIST, Uniform-Random, Gaussian-Random as the test OOD datasets. Compare the performance w/ and w/o OE. Does OE bring in comparable improvement for all OOD datasets?
- Evaluate OE's performance v.s. diversity of the outlier dataset. Here we measure diversity by the number of classes contained in the outlier set. Train multiple OE models using the first 1, 5, 20, 50, 100 classes. How does the diversity affect results?

References:

[1] Hendrycks and Gimpel, "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks," (https://arxiv.org/abs/1610.02136)

[2] Liang et al., "ENHANCING THE RELIABILITY OF OUT-OF-DISTRIBUTION IMAGE DETECTION IN NEURAL NETWORKS", (https://arxiv.org/pdf/1706.02690.pdf)

[3] Liu et al., "Energy-based Out-of-distribution Detection," (http://arxiv.org/abs/2010.03759)

[4] Hendrycks et al., "Deep Anomaly Detection with Outlier Exposure," (https://openreview.net/forum?id=HyxCxhRcY7)

Project 9: Self-Supervised / Representation Learning

Lead TA: Jingyang Zhang

Project description:

Self-Supervised / Representation Learning is all about learning useful information from **unlabeled** data. The key is that the supervisory signal comes from the data itself, without the need for manual labeling or annotations. One method for extracting useful information from unlabeled data is to perform contrastive learning, which works by maximizing "agreement" between model output for augmented versions of the same sample. In this project, you will implement the SimCLR framework for contrastive learning of representations on CIFAR-10, then show how the model can be trained as a classifier using only a fraction of the labels.

Requirement:

- Implement the SimCLR [1] method for the CIFAR-10 dataset. See Appendix B.9 of the paper for detailed setup on CIFAR-10.
- Pick a reasonable batch size and number of training epochs for your hardware platform, and perform a linear evaluation to reproduce the corresponding result from Figure B.7.
- Evaluate the effect of having less/limited labeled data. Follow Section 6 – Semi-supervised learning and Table 7 for inspiration of how to set this up on CIFAR10. Compare your SimCLR model to the "supervised baseline". Is the SimCLR model better suited when there are limited amounts of labeled training data?
- Implement the RotNet [2] learning framework and compare/contrast its performance to SimCLR. Use the same two experiments described in the basic requirements for comparison.

References:

[1] Chen et al., "A Simple Framework for Contrastive Learning of Visual Representations," (https://arxiv.org/pdf/2002.05709.pdf)

[2] Gidaris et al., "Unsupervised Representation Learning by Predicting Image Rotations", (https://arxiv.org/abs/1803.07728)

Project 10: Adversarial Patch Attack

Lead TA: Jingyang Zhang

Project description:

Previously we have studied the adversarial attacks, which are realized by crafting pixel-wise additive perturbations that are constrained by L-p norm bound. These attacks, as you may notice, are not that feasible in the real-world (can you really add small noises to a physical object?). In contrast, there is another type of adversarial attacks that are designed to be “physical” – Adversarial Patch attack. As its name suggests, the idea is to create a patch (or a sticker) that can be attached to somewhere in the image to fool the classifier. In this project, you will act as an attacker and craft such adversarial patches to mislead neural networks.

Requirement:

- Implement the Adversarial Patch generation process described in [1] for CIFAR-10 classifiers. For simplicity you can generate rectangular patches instead of circular ones.
- Evaluate the effect of patch size. For example, generate patches with the size 3x3, 5x5, 7x7, 16x16, and measure the *untargeted* attack success rate (ASR) as a function of the patch size. Are there any visual patterns you could see in the generated patches?
- A careful attacker may want to disrupt the model in a way that whenever the patch is there the model will predict a certain target class. Repeat step 2 but this time focus on the *targeted* attack success rate. Try using each of the 10 classes as the target class. How does targeted ASR compared to untargeted ASR? Are there any visual patterns when generating targeted patches?
- Transfer the generated patches on one model to others, e.g., from ResNet to DenseNet or VGG. Measure the untargeted/targeted ASR. To what extent do the patches transfer across different models?

References:

[1] Brown et al., “Adversarial Patch,” (https://arxiv.org/pdf/2002.05709.pdf)

Project 11: Image Synthesis with A Single (Robust) Classifier

Lead TA: Jingyang Zhang

Project description:

The fantasy of deep learning is not only that it can perform discriminative tasks like classification and detection, but also that it can do generative jobs! The conventional wisdom was that you have to train models explicitly in a generative way such that they can tackle generative tasks (consider how Generative Adversarial Networks are trained). However, recently researchers found that actually a single adversarially robust model is enough. In this project, you will implement and perform several exciting image synthesis tasks like image generation, inpainting, super-resolution (and image-to-image translation if computation resource allows).

Note:

- Thanks to the authors of [1] who released all their pre-trained models (https://github.com/MadryLab/robustness_applications?), you don't need to train the adversarially robust models yourself (which could be super time-consuming). For the project, you could use images from either CIFAR-10 or ImageNet, depending on the capacity of your computing environment (e.g., whether it can stores the large ImageNet dataset and whether the GPUs have enough memory). If possible, I strongly recommend that you use ImageNet which has high-resolution images and makes everything fancier than using the low-resolution CIFAR images. Also, this whole project can be a playground for you. It is encouraged that you do more than the basic requirements listed below based on your interest.

Requirement:

- Implement image generation (Sec. 3.1). If you use CIFAR-10, generate 5 images for each of the 10 classes. If you use ImageNet, generate 3 images for 10 classes that are of your interest. You can use either random noises or a seed image from another class as initialization. Do the synthesized images present clear visual patterns? Also try the same thing using a standard, non-robust classifier. Can the non-robust classifier work as well?
- Implement image inpainting (Sec. 3.2). If you use CIFAR-10, randomly pick 5 images from each of the 10 classes to inpaint. If you use ImageNet, randomly pick 3 images from 10 classes that are of your interest to inpaint. Do the inpainted images look natural/plausible?
- Implement image super-resolution (Sec. 3.4). If you use CIFAR-10, randomly pick 1 image from each of the 10 classes, downsample them to 16x16, and perform the super-resolution. If you use ImageNet, randomly pick 1 image from 10 classes that are of your interest, downsample them to 64x64, and perform super-resolution.
- Optional but strongly encouraged: Implement the image-to-image translation (Sec. 3.3). You can download the pre-trained models again via the github link above. Choose several source-target pairs (maybe 3 or more) in a similar fashion to Figure 5 and perform image-to-image translation.

References:

[1] Santurkar et al., "Image Synthesis with a Single (Robust) Classifier," (https://arxiv.org/pdf/1906.09453.pdf)

Project 12: Neural Network Distillation

Lead TA: Meng Xia

Project description:

In class, we have learned various compression methods to compress Neural Networks. In this project, we will investigate another way to build light weight deep networks commonly known as distillation [1][2][3]. The main idea is to let a small model, usually referred to as student model, mimic the predictions of a large model, usually referred to as teacher model. Experiments have revealed that this usually leads to better performing student models compared to solely training from ground truth labels.

Requirement:

- Design a small network and a large network by yourself following the instructions on section 3 in [1]. Report the FLOPS and number of parameters of both models then train them on MNIST and report their performance (accuracy) on validation dataset.
- Use the soft labels generated by the larger model to train the small network on MNIST and compare the performance with the same small model trained with ground truth.
- Try different temperature T in equation (1) in [1] and analyze the influence of this parameter on distillation.
- We refer to the dataset which the large model extracts soft label of, and the small model is trained on as transfer dataset. Obviously, the transfer dataset need not be identical to the training set of the large model. Now try to randomly omit one digit from MNIST as transfer set and repeat the distillation process (large model is still trained on full MNIST). Test the trained small model on complete validation dataset to see what the performance of this small model on the digit that it never sees during training.
- Then, try to reverse the teacher and the student model [2], which means now the student model is larger than teacher model. For this question, you could pick up two existing models by yourself as teacher and student models, such as ResNet34 and ResNet50 and experiment on Cifar10 dataset.
- Instead of learning from a teacher model, model can also learn from itself for improving performance, which is known as self distillation. Implement self distillation for ResNet50 on Cifar10 based on section 3 of [2]. Then compare the performance between traditional training model and model trained with self distillation.
- Based on your experiments results, could you provide several reasons for why distillation can improve the performance of neural networks?

References:

[1] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." arXiv preprint arXiv:1503.02531 (2015).

[2] Yuan, Li, et al. "Revisit knowledge distillation: a teacher-free framework." (2019).

[3] Zhang, Linfeng, et al. "Be your own teacher: Improve the performance of convolutional neural networks via self distillation." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019

Project 13: Dynamic BlockDrop

Lead TA: Meng Xia

Project description:

Not all images are equally difficult to recognize. It is intuitive that some simple images can be correctly classified with a smaller model, but the difficult ones have to be classified by a larger model. Based on this motivation and the observation that ResNets behave like ensembles [1], authors of [2] proposed to learn a per-instance network routing policy that will adaptively drop certain blocks in a neural network for each image, potentially saving computational cost for the samples deemed to be ‘easy’ with minimal sacrifice of the overall accuracy. In this project we will verify the ensemble-like property of ResNets, and try to accelerate by exploiting this property.

Requirement:

- Train a VGG network on Cifar10 then randomly drop different layers of it and report the influence of dropping different layers on performance.
- Repeat the above experiment but this time, instead of using VGG network, try to use ResNet34 and report the influence of dropping different layers on performance (Hint: the influence of downsampling layers is different with other layers.)
- Now, we will try to drop as much as possible layers of residual networks for each image while not loss much performance. Implement a dynamic drop policy based on section 3 of [2] for ResNet34 on Cifar10.
- Test your final dynamic drop policy for ResNet34 on Cifar10. Then compare randomly drop policy and dynamic drop policy based on dropping ratio (how many blocks you drop) and performance (accuracy on validation dataset.)
- Based on your results, could you provide several reasons for explaining why residual networks are better than it previous networks, such as VGG?

References:

[1] Veit, Andreas, Michael J. Wilber, and Serge Belongie. "Residual networks behave like ensembles of relatively shallow networks." Advances in neural information processing systems 29 (2016): 550-558.

[2] Wu, Zuxuan, et al. "Blockdrop: Dynamic inference paths in residual networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

Project 14: Generative Adversarial Networks

Lead TA: Meng Xia

Project description:

Most of the techniques we discuss in this course are “discriminative” in nature, where the models are trained to make decisions with accuracy. On the other hand, ”generative” networks are trained to model the training distribution as to be capable of generating new samples from it. In this project you will create and train two Generative Adversarial Network variants: conditional Generative Adversarial Network (GAN) called the Auxiliary Classifier GAN (AC-GAN) [1] and Wasserstein GAN[2]. You will firstly implement the AC-GAN for the CIFAR-10 dataset, then show its ability to generate new, never-before-seen samples from CIFAR-10’s underlying data distribution. Then, you will implement the WGAN and investigate its ability to make GAN training process stable.

Requirement:

- Implement both the traditional GAN based on section 3 of [1] and WGAN based on section 3 of [2] for CIFAR10 dataset. Qualitatively evaluate the performance of your GAN and WGAN. Compare the training process between WGAN and the original GAN to see which one is more stable.
- Implement an Auxiliary Classifier GAN (AC-GAN) [1] for the CIFAR10 dataset. Follow the architecture and training procedure described in Table 2 of the paper.
- Qualitatively evaluate the performance of your GAN by generating several new samples from each class and commenting on their quality.
- Perform latent space interpolations between some samples from the CIFAR10 test-set and comment on the implications of such interpolations (See Fig. 9 of paper).
- The traditional Generative Adversarial Networks are usually unstable and hard to be trained. People [3] thought this problem caused by the loss function of original GAN [2], so they provided a new loss function for making GAN training more stable. Implement the WGAN based on section 3 of [3] for CIFAR10 dataset. Compare the gradients vary between WGAN [3] and the original GAN [2] to see which one is more stable.

References:

[1] Odena, Augustus, Christopher Olah, and Jonathon Shlens. "Conditional image synthesis with auxiliary classifier gans." International conference on machine learning. PMLR, 2017.

[2] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems 27 (2014).

[3] Arjovsky, Martin, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks." International conference on machine learning. PMLR, 2017

Project 15: Self-Supervised Learning for Language Modeling

Lead TA: Yuan Li

Project description:

Self-supervised learning (SSL) has gained a lot of attention in the natural language processing (NLP) community because it can effectively learn from large-scale raw data which can be obtained without any expensive manual labeling, and enables models to be scaled to millions of parameters. Models pre-trained through self-supervised learning usually demonstrate strong versatility and can be effectively transferred to various downstream tasks. One of the most important components of self-supervised learning is the pretext tasks, which encourages the model to learn general knowledge by exploiting the data itself including its context and structure. BERT [1] is one of the most popular and classic pre-training models. It demonstrates powerful transfer capabilities for many NLP tasks, including question answering, text classification, and sentiment analysis. BERT employs two pretext tasks: masked language modeling (MLM) and next sentence prediction (NSP). The goal of this project is to explore the effectiveness of different pretext tasks for language modeling. You will gain hands-on experience on BERT's pretext tasks and tasks in other models such as ALBERT [2] and BART [3].

Requirement:

- First, use HW3 as the starting point for this project. Implement a 3-layer Transformer encoder and choose your own hyperparameters (as long as the model does not exceed the memory limit). Train this model to perform sentiment analysis on the IMDB dataset. Use this model as the baseline for the entire project.
- Implement MLM and NSP pretext tasks. For more details, see BERT [1].
- Implement sentence order prediction (SOP) pretext task. For more details, see ALBERT [2].
- Implement the sentence permutation (SP) pretext task. For more details, see BART [3].
- Use these pretext tasks to pre-train the model, and then fine-tune the pre-trained model on the downstream task. Obtain the prediction accuracy of the pre-training stage (for example, the accuracy of masked token prediction, next sentence prediction, sentence order prediction, and sentence permutation prediction), and the accuracy of sentiment classification when evaluating on the downstream task.
- Perform ablation studies for each pretext task implemented (for example, compare the results of each individual task and its combination).

References:

1. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." (https://arxiv.org/pdf/1810.04805.pdf)
2. Lan, Zhenzhong, et al. "Albert: A lite bert for self-supervised learning of language representations." (https://arxiv.org/pdf/1909.11942.pdf)
3. Lewis, Mike, et al. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension." (https://arxiv.org/pdf/1910.13461.pdf)

Project 16: Learned Non-linear Quantization

Lead TA: Congjia Yu

Project description:

As introduced in the lecture, a non-linear quantization scheme can reach a higher performance comparing to a linear quantization. In this project, you will be implementing one of the state-of-the-art non-linear quantization algorithms, LQ-Nets [1] (ECCV 2018), and observe its performance under different precision

Requirements:

- Read the LQ-Net paper and implement the quantizer and network training algorithm proposed in Section 3.3 in PyTorch.
- Use the implemented algorithm to train ResNet-20 models on CIFAR-10, with weights quantized to 3-bit, 2-bit and 1-bit. Compare your results with the results reported in Table 4.
- Apply the same quantization scheme to the activations, quantize the activations to 2-bit or 3-bit, compare the results with those reported in Table 4.
- Visualize the final weight distribution of each layer before and after quantization. How is the distribution different from linear quantization?
- Add sparsity-inducing optimization technique (e.g., soft thresholding) to the basis update step in LQ-Net algorithm, achieving mixed-precision quantization schemes by inducing zeros in the basis vector. You can only consider weight quantization for this part.
- Tune the strength of the sparsity-inducing optimization and observe the tradeoff between accuracy and average precision. How is it compared with the original LQ-Net performance?

References:

[1] Zhang et al., “LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks,” (https://arxiv.org/pdf/1807.10029.pdf)

Project 17: Compressing the Transformer Model

Lead TA: Congjia Yu

Project description:

With the prevalence of Transformer models and the attention mechanism, neural machine translation tasks witness a huge improvement. Transformer model [1] uses stacked self-attention and point-wise, fully connected layers for both the encoder and decoder. However, all these layers contribute to millions of parameters consumption, which is too demanding for Transformer models to be deployed onto memory-constrained systems. In this project, you will learn the structure of Transformer models and try to apply model compression techniques, such as pruning and quantization, to make the transformer model more compact and friendly for memory-constraint systems. You are also encouraged to apply additional techniques (e.g., low-rank factorization, sparsity-induced regularization, etc.) to enforce a higher compression rate on Transformer models. Note that all experiments are conducted on the WMT'14 English-German translation benchmark.

Requirements:

- Read and understand the Transformer paper. Try to conduct compression on the Transformer architecture using all the techniques you learned, including pruning, quantization, sparsity-induced regularizations etc. Try to obtain a high compression rate for Transformer models.
- Try to change the structure of Transformer models by changing some architecture hyperparameters (i.e., ff, d_model) and obtain transformer models with different sizes. Observe the effect of model compression on different-sized models and find out the trade-off between effective parameters (i.e., original parameters * compression rate) and performance (i.e., BLEU score). Make a conclusion on what is the best way to obtain an efficient Transformer model.
- Try some advanced model compression schemes, including mixed-precision quantization [2] and low-rank factorization [3]. You can also automate the process of mixed-precision quantization by referring to [4].
- Attention layers and feed-forward layers have different semantic meanings in NLP. Study the difference when you apply model compression techniques. Make a conclusion on which type of layer is easier to compress.
- Apply model compression techniques on the word embedding layer and report your findings.

References:

[1] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.

[2] Wang, Kuan, et al. "HAQ: Hardware-Aware Automated Quantization with Mixed Precision." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[3] Denton, Emily L., et al. "Exploiting linear structure within convolutional networks for efficient evaluation." Advances in neural information processing systems. 2014.

[4] Cheng, Hsin-Pai, et al. "Differentiable fine-grained quantization for deep neural network compression." arXiv preprint arXiv:1810.10351 (2018).

Project 18: Poisoning attack to Federated Learning

Lead TA: Hongyu He

Project description:

In federated learning, multiple client devices jointly learn a machine learning model. Each client device maintains a local model using its local training dataset, while a master device maintains a global model by aggregating the local models from the client devices. Recent research has shown that standard federated learning techniques are vulnerable to poisoning attacks, where the attacker can manipulate the local training datasets or the local model parameters during the learning process such that the global model has a large testing error rate. In this project, we aim at exploring the vulnerability of federated learning against such poisoning attack and discuss the potential defense methods.

Requirements:

- Implement federated learning on one of the following datasets: MNIST, Fashion-MNIST, CH-MNIST, CIFAR-10, ImageNet following the method proposed in [1].
- There're two kinds of poisoning attacks: data poisoning attacks and model poisoning attacks. What's the difference between these two poisoning attacks? In other words, what an attacker knows and what an attacker can do in these two different attacks?
- Implement data poisoning attack[2] or model poisoning attack[3] to the federated learning you have implemented. Analyze the results.
- If a model is robust against data poisoning attacks, is it also robust against model poisoning attacks? Which kind of attack do you think is stronger?
- Discuss ways to defense against poisoning attacks.

References:

[1] Communication-Efficient Learning of Deep Networks from Decentralized Data

[2] Data Poisoning Attacks on Federated Machine Learning

[3] Local Model Poisoning Attacks to Byzantine-Robust Federated Learning

Project 19: Deepfake Generation

Lead TA: Hongyu He

Project description:

As deep learning becomes more and more powerful, deep learning based deepfake generation methods can produce more and more realistic-looking deepfakes. Fake face images can be generated by either face synthesizing or face manipulation. Face synthesizing creates new faces that do not belong to any person in the world, while face manipulation manipulates the face images of existing individuals. Generally speaking, face manipulation may result in more severe consequences than face synthesizing, because it may target at individuals who have high social impact or make illegal use of other people's faces. Therefore, in this project, we will focus on face manipulation based Deepfake generation and discuss the potential ways to defense against it.

Requirements:

- Prepare datasets for Deepfake generation. Here's some typical datasets for you to consider: FaceForensics++[1], DFDC[2]. Feel free to search for other datasets. You can also use your own face images and your friends' (should be fun ^_^).
- Face manipulation can be further categorized into face replacement (face swap) and face reenactment. In face replacement, a target face is replaced as a source face and the generated fake face claims to be the source. In face reenactment, the expression, hair color, and/or other properties of the target face are changed as those of a source face, but the identity of the target face is still preserved. Implement both face replacement and face reenactment. Show the fake faces you generated and comment on the results.
- Existing deepfake detection mainly focuses on passive detection. Train a classifier using the true faces and fake faces you generated to detect whether a given face is deepfake. Try many different classification models, compare their performances and select the best one. Select a proper metrics to measure the performance of the classifier and report the results.
- Use another dataset to train your deepfake models and generate fake faces. Then use the classifier you trained before to detect true/fake faces (Do not re-train the classifier on new fake faces). Can this classifier still correctly identify the fake faces?
- Do you think it's a good way to defense against deepfake? Discuss different methods to defense against the deepfake or reduce the harm of deepfake.

References:

[1] FaceForensics++: Learning to Detect Manipulated Facial Images

[2] The DeepFake Detection Challenge (DFDC) Dataset

Project 20: Decision-based Black-box Adversarial Attacks

Lead TA: Hongyu He

Project description:

In a very realistic blackboxadversarial attack setting, the adversary may not have access to the training dataset of the target model, which rules out transfer attacks. Further, the target model may only return the predicted class(not the full probability vector over all classes) and has a built-in defense limiting the number of successive queries that can be made by a single user. Using only the predicted class information, decision-based attacks function by walking along the decision boundary of the classifier until coming within the vicinity of a known sample from the target class, then taking one final step into the classification region of that class. In this project, you will implement a recent and state-of-the-art decision-based attack called the “HopSkipJumpAttack” (HSJA) [1], and measure its performance in terms of number of queries and size of perturbation to achieve attack success.

Requirements:

- Implement the L2 version of HSJA and reproduce the untargeted attack results in Table 1 (of [1]) for the ResNet-CIFAR10 case
- Analyze the untargeted attack by recreating two of the analysis from the HSJA paper. Specifically, pick between Figures 3, 4, 5, 6, 7 and reproduce two of them for the ResNet-CIFAR10 case.
- Implement the “Biased Boundary Attack” (BBA) [2] and compare its untargeted attack performance to the HSJA for the ResNet-CIFAR10 case
- Compare the BBA’s performance to the HSJA in terms of number of queries, size of perturbation required, and adversarial success rate
- Perform the same analysis you did for the HSJA attack
- Make conclusion: Which attack is better? Why?

References:

[1] Chen et al., “HopSkipJumpAttack: A Query-Efficient Decision-Based Attack,” (https://arxiv.org/pdf/1904.02144.pdf)

[2] Brunner et al., “Guessing Smart: Biased Sampling for Efficient Black-Box Adversarial Attacks,” (https://arxiv.org/pdf/1812.09803.pdf)