Marvel Phase 5: The Convengers Initiative

Sean Lin
seanlin2000@berkeley.edu

Norman Karr
nkarr11@berkeley.edu

1 Introduction

In this computer vision project, our primary goal was to build a robust image classifier on Tiny ImageNet. We aimed to build and train our classifiers in such a way that they would be resistant to various data perturbations, adversarial examples, or out-of-distribution data. The importance of this specific task cannot be understated because setting up models to be robust is a key step in ensuring good real-time performance as well as defending against adversarial attacks.

To tackle this problem, we opted to use a transfer learning approach where we use pretrained networks and fine-tune the networks so that they are optimized for our task. In this report, we introduce two new forms of transfer learning models: multi-model ensembling and multi-model feature extraction dubbed Convengers. We evaluated each model using top-1 and top-5 accuracy metrics on two splits of data: a clean validation set and a adversarial validation set. Thus far, we have been able to achieve good results with Convengers with 70% top-1 accuracy, 89% top-5 accuracy, and 48% top-1 accuracy on adversaries. Convengers performed roughly equal to our baselines for clean data but performed noticeably better accuracy and adversarial accuracy. Regardless, the results of both models look promising but could largely benefit from more training.

2 Literature/Related Work

2.1 Adversarial Robustness

Adversarial attacks have long been intriguing in machine learning as a way to fool seemingly good classifiers. Goodfellow et al. (2014) (1) invented the fast gradient sign method (FGSM), which generates adversarial methods by adding or subtracting a small ϵ to each pixel of the image, depending on the sign of its gradient. This is done to intentionally alter the image to fool the model. While seemingly simple, this type of adversarial attack proves to work very well on most models.

Madry et al. (2019) (2) devised the Projected Gradient Descent (PGD) attack, which attempts to find the perturbation that maximizes the loss of the model on the input via constrained optimization. Because it seeks to solve an optimization problem rather than taking iterative steps like FGSM, adversarial examples generated by PGD takes significantly longer to generate. However, the adversarial attacks that PGD generates are much stronger, as they are optimized to find the adversarial example that maximizes loss. PGD attacks are empirically difficult to defend against.

Although adversarial attacks are difficult to defend against, there is some research on defenses. Goodfellow et al. (2014) (1) suggested training on adversarial examples as well as clean examples to increase model robustness. Liao et al. (2018) (3) suggested preprocessing the input, such as with median smoothing filters or bit squeezing, to make adversarial images more "clean." Papernot et al. (2016) (4) has even suggested using knowledge distillation as a defense against adversarial attacks, as knowledge distillation would distill the gradients used in adversarial sample creation. Our work focuses on adversarial training as a means to adversarial robustness.

2.2 U-Net

While not directly implemented in our project, U-Net was a key source of inspiration for our final model. Developed by Ronneberger et al. (5), U-Net consists of a contracting path of convolutional

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

layers to capture important features in the data, and then upsamples using transpose convolutions to get the output segmentation map. On the expanding path, U-Net concatenates the features from the corresponding convolutional layer to the current layer's features, which provides additional information that improves its performance. This idea of concatenating features from a different source was the source of inspiration for our Convenger Initiative.

3 Background

3.1 Pre-trained Models:

In both our networks, we build off three different pre-trained models: ResNet-101 (6), Inception-V3 (7), and VGG-19 with batch normalization (8). We specifically chose these three models because they each have significantly different architectures while being trained on the same ImageNet classification task.

Inception was chosen because it's "inception" blocks, which apply differently sized filters, would be able to capture features invariant on the relative size of objects in frame. VGG-19 was the first computer vision architecture to utilize homogenous stacks of multiple convolutions interspersed with resolution reduction. It doubled the depth of AlexNet, and provides 4096 features post-convolutional processing, which is double that of Inception or ResNet. We hypothesized that this larger number of features may be useful for feature extraction during transfer learning of this task. By adding skip connections to improve gradient flow, ResNet-101 was able to achieve much larger depth than VGG, having 101 layers rather than 19, without suffering from gradient saturation. Therefore, ResNet features may be far more processed than VGG's, and may be useful for our feature extraction.

We expect that combining these models will benefit overall accuracy as well as out-of-distribution performance because we are no longer confined to a single model's featurization or decision. How we actually utilize the pre-trained models differs between our two architectures and will be discussed in the methods section.

3.2 Transfer Learning Transfer learning takes a pre-trained model and optimizes it for a new task by attaching a linear classifier to the end that is trained using one of two methods: fixed feature extraction and fine tuning. Fixed feature extraction is when the pre-trained parameters are fixed and only the new layers are trained, whereas fine-tuning is when all parameters are trained. These methods can also be performed sequentially (referred to as two-phase training). Each of these methods are commonly used, but we performed some basic tests on a simple ResNet transfer model and found that fine-tuning outperformed two-phase training and fixed feature extraction.

4 Methods

4.1 Data Augmentation

Generally, data augmentations are necessary because they help make networks more robust to transformations and noise. In our case, they were imperative because our images were 64x64 pixels while the pre-trained models take 224x224 pixel images as input. To combat this, we resize all images to a higher resolution of 256x256 pixels using bi-linear interpolation and then a random 224x224 crop is made. Originally, we wanted to upsample our images using transpose convolutions, but opted for interpolation because this method does not require training parameters. In addition to resizing, we include also include a random Gaussian blur to each training image with probability 0.25. By blurring some training examples, our model can develop robustness towards blurred images, which may appear in the test set. Lastly, we normalize the pixels across each channel based on the same standards that pre-trained models used.

4.2 Adversarial Training

Because adversarial training is a proven defense against adversarial attacks, we decided to implement this method of training. The two adversarial attacks we opted to defend against were FGSM and PGD. Since both of these attacks are gradient-based, we constantly generate new adversaries by running FGSM and PGD attacks on a clean mini-batch of training data to produce a mini-batch of adversarial data . This process is then repeated for all iterations.

4.3 Baseline Models

We use individually transfer-learnt ResNet-101 and Inception-V3 models as our baseline. In the ResNet baseline, the fully connected layer at the end is replaced with a singular fully connected layer with an output dimension of 200. A similar adjustment was made to the Inception baseline' however, in Inception we also made sure to ignore the built-in auxilary classifier. Both of these baselines were trained once without adversarial training and once with adversarial training. While we also attempted to create a VGG baseline, VGG proved incredibly difficult to train, and converged with significantly poorer accuracy than the other two models. This is consistent with the paper on VGG, which describes the lengthy VGG training process of training smaller versions of VGG before training the whole model. As a result, the baseline models for the experiment were transfer-learnt ResNet-101 and Inception-V3.

4.4 Multi-model Ensembling

Our first novel architecture is an ensemble of models of different architectures, which we dub *multi-model ensembling*. The ensemble consists of 3 transfer-learnt ResNet-101 classifiers and 3 transfer-learnt Inception-V3 classifiers. We hypothesized that by ensembling two different model architectures rather than many of the same model architecture, the ensemble may be more resistant to adversarial examples. Even though we have seen in lecture that adversarial examples can transfer across models, we hypothesize that their means of counteracting the adversaries post-adversarial-training may be different. Therefore, an ensemble of two different model architectures may be more robust against generated adversaries.

Each of the 6 basic models making up our ensemble were each trained separately. Each one was trained using Adam for 20 epochs. For the learning rate, we began the parameters we determined from our baselines and then used a learning rate scheduler to decrease the learning rate by a factor of 10 every 7 epochs.

4.5 Convengers Initiative

Model Architecture

Inspired by U-Net's concatenation of features from previous layers to provide additional information for improved performance, we came up with our own novel combination architecture: concatenating extracted features from the three of the top performing computer vision models in industry—ResNet-101, Inception-V3, and VGG-19—to develop a robust classifier. Dubbed the Convengers Initiative, we connected the extracted features of these three models together with dense layers as one singular model. For both ResNet and Inception, we flattened the output of each model's final average pool layer to get two 2048 dimensional vectors as our feature vectors. For VGG, we chose to extract VGG features from the output of its first fully connected layer which had an input dimension of 25088 and output dimension of 4096. This way we could fine-tune the 100 million parameters instead of having to train them from scratch.

With these extractions, we have 3 resultant features vectors for ResNet, Inception, and VGG with respective dimensions of 2048, 2048, and 4096. Using these three vectors, we then concatenate them into one large 8192-dimensional feature vector. We then connected this vector to 3 blocks (each block consists of a dense layers each with batch norm, ReLU, and dropout) and then one final dense layer to produce the final logits.

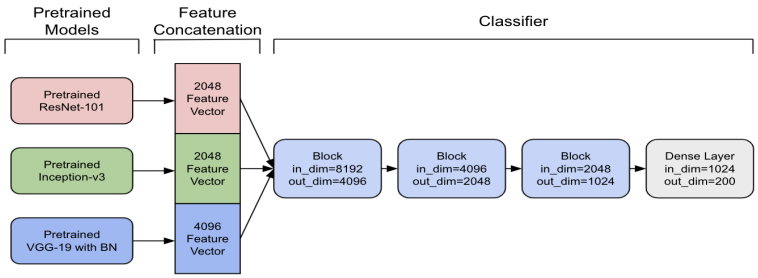

Figure 1: Convengers Architecture

5-Phase Training

We are also introducing a novel training method for this architecture which we dub Marvel Phase 5, a 5-phase training pipeline. For each phase, we use the Adam optimizer and load data with a batch size of 64. Additionally, all phases except for phase 1 are also trained with adversarial examples with a 10:1 ratio of FGSM adversaries to PGD adversaries.

In the first phase, we hold the parameters of the pre-trained layers constant and train only the newly added layers. The idea is that hopefully this will help our dense layers gain insight about the features in the pre-trained models. The next three phases are then trained where each phase fine-tunes the new layers plus one of the pre-trained models. For example, phase 2 trains the ResNet and dense layers while the Inception and VGG layers are frozen. Each of these phases also have different learning rates which we determined from earlier tests with the respective baselines. If we had more time and compute, we wanted to loop phase 2-4 so that it resembles something like a clustering algorithm. In the final phase, we unfreeze all layers and fine-tune the model end-to-end for 15 epochs using a learning rate scheduler starts at $1e^{-5}$ and decreases by a factor of 10 every 7 epochs.

5 Results

All relevant models that we trained with adversarial training were trained on Colab and AWS spot instances. In the end, we were able to successfully train to completion a baseline for both ResNet and Inception. Unfortunately, both the ensemble and Convengers were not able to train to completion. We ran into problems with Colab saying we exceeded our limit of GPU compute or AWS terminating our spot instances prematurely. For the ensemble, we were only able to fully train two of the six models included in the ensemble. For Convengers, we were able to complete the first 4 phases of the five phase train and then only a few epochs of the final end-to-end training phase. For both cases, it was clear that we had not yet fully converged. Nevertheless, the data that we were able to is displayed below.

Accuracy					
Model	Top-1 Train	Top-5 Train	Top-1 Val	Top-5 Val	Top-1 Val (w adv.)
ResNet	64.75	83.75	77.06	85.75	47.31
Inception	64.69	82.50	71.94	89.18	46.12
Ensemble	51.20	75.76	57.21	77.19	–
Convengers	65.89	81.72	70.56	88.74	48.61

One immediately surprising observation is that the training accuracy for all models was less than the validation accuracy. However, we deduced that this may be due to the fact that the data augmentations performed on training data was different than validation data. As a result, we believe that the resulting training data was a more difficult problem for the networks. Despite not being trained to completion, it is clear that Convengers has promise as an architecture. It was able to achieve similar accuracy across each metric and actually performed better in adversarial validation. The observation follows that of the trade-off of optimizing for adversaries suggests that our model is actually building resistance to attacks as it should.

6 Conclusion

We were surprised by how well Convengers performed on the data, because we hypothesized that the model may be extremely difficult to train. VGG, Inception, and ResNet were all trained using different hyperparameters—such as learning rate schedulers, momentum weights, etc.—so we thought it may be difficult to train all of their backbones together in one model. We were also concerned that Convengers may overfit. Often times, models that have too many parameters tend to overfit to the training data, and perform poorly on the validation set. Given the potential for overfitting or training issues, we were pleased with the results, to say the least.

7 Team Contributions

Both of us did an equal amount of work. In exact percentages, we split the work 50-50. We worked together on team calls from 8pm to 12:00am every day of dead week and the days of finals week leading to the Wednesday deadline. We also worked individually for roughly an hour every day prior to meeting up to work as a group. Below we have detailed our individual contributions to the project:

Sean Lin: Researched how to implement transfer learning and tested whether it would be more effective to use fine-tuning or feature extraction for this project. Set up our AWS instances and data pipeline, which involved curling the dataset into the proper directory and writing a preprocessing utils file to properly preprocess the validation set to remain consistent with the training set. Wrote the linear classifier that we attach to the back of pre-trained models for transfer learning, and wrote the naive version of our model trainer. Fine-tuned our ResNet models and tuned the ResNet hyperparameters, including batch size, learning rate, and learning rate scheduler. Developed the multi-model model ensembling method, which includes a ensemble of ResNet-101 and InceptionV3 models. Researched adversarial examples, plotted a small batch of examples to test our baseline model with, and implemented adversarial training.

Norman Karr: Wrote a set of utils functions to correlate image class number with the actual name of the image class. Analyzed the existing architecture of each of the three pretrained models in pytorch for implementing the transfer learning models and developed the code for each model as well. In addition to developing the models, Norman also built a class for training and other important functionality such as saving models and model logs. In terms of model testing, Norman mainly tested Inception-V3 and VGG-19 transfer models. The Convenger initiative was primarily completed by Norman as he implemented the concatenated model, came up with the five phase training method, and trained the model.

8 Appendix

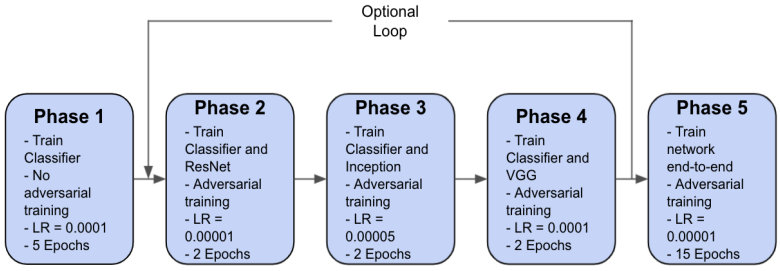

Figure 2: Five Phase Training

References

[1] G. et al., "Explaining and harnessing adversarial examples." https://arxiv.org/pdf/1412.6572.pdf, March 2015.

[2] M. et al., "Towards deep learning models resistant to adversarial attacks." https://arxiv.org/pdf/1706.06083.pdf, September 2019.

[3] L. et al., "Defense against adversarial attacks usinghigh-level representation guided denoiser." https://openaccess.thecvf.com/content_cvpr_2018/papers/Liao_Defense_Against_Adversarial_CVPR_2018_paper.pdf, 2018.

[4] P. et al., "Distillation as a defense to adversarialperturbations against deep neural networks." https://arxiv.org/pdf/1511.04508.pdf, 2016.

[5] R. et al., "U-net: Convolutional networks for biomedicalimage segmentation." https://arxiv.org/pdf/1505.04597.pdf, 2015.

[6] H. et al., "Deep residual learning for image recognition." https://arxiv.org/pdf/1512.03385.pdf, December 2015.

[7] S. et al., "Rethinking the inception architecture for computer vision." https://arxiv.org/pdf/1512.00567.pdf, December 2015.

[8] S. et al., "Very deep convolutional networks for large-scale image recognition." https://arxiv.org/pdf/1409.1556.pdf, April 2015.