Developing Robust Networks to Defend Against Adversarial Examples

Benson Liu
University of California, Los Angeles
bensonhliu@gmail.com

Isabella Qian
University of California, Los Angeles
iqian@ucla.edu

Abstract

An ongoing area of ML research involves defenses against adversarial examples, which are intentionally designed inputs that attack a model's inference. We explore the impact of adversarial examples against common deep learning classification architectures such as CNNs and RNNs, specifically testing on CNN and CNN-LSTM hybrid models trained on the CIFAR-10 dataset. We implemented an efficient adversarial attack known as the Fast-Gradient Sign Method (FGSM) against the baseline model. We then developed more robust networks specifically to defend against adversarial attacks by performing data augmentation on the original dataset with adversarial examples. We tested multiple robust models this dataset. Our baseline models had a 80.374% accuracy for the CNN and 79.001% for the CNN-LSTM hybrid architecture, and when attacked with adversarial examples with $\epsilon = 0.1$, their accuracies dropped to 9.375% and 12.679%, respectively. In the same adversarial tests, our best robust models achieved a validation accuracy of 75.488% for the CNN and 74.056% for the CNN-LSTM hybrid, which are absolute increases of 64.142% and 61.375% from the base models. When comparing architectures, although there were strengths to the CNN-LSTM hybrid architecture, the CNN outperformed it in terms of accuracy and training time for adversarial examples.

1. Introduction

In recently years, AI image recognition has become increasingly versatile, but also poses new risks. By altering input data in a way undetectable to humans, machine learning models can be fooled into incorrect classification. These are known as *adversarial examples*. In this paper, we explore the scope of generation adversarial example as well as a method for defending against these kinds of attacks using a form of data augmentation known as adversarial training. We particularly explore these on computer vision models for the CIFAR-10, which is a dataset of 60,000 32x32 color images in 10 classes.

1.1. Fast Gradient Sign Method (FGSM)

First introduced by *Goodfellow et al*, the Fast Gradient Sign Method (FGSM) is a computationally efficient and easy to implement algorithm for generating adversarial examples [1]. The given threat model is a white-box attack where the attacker has access to both the training data and the underlying model. This technique involves computing the gradient of the loss function with respect to the input data and then adding a perturbation to the input data in the direction of the gradient that maximizes the loss. The perturbation η can be calculated to the following formula where ϵ is a perturbation constant:

$$\eta = \epsilon \text{sign}(\nabla_x J(\theta, x, y))$$

While other methods for generating adversarial examples exists such as Jacobian-based Saliency Map Approach (JSMA) or Carlini and Wagner Attack (C&W), these are often more computationally expensive as they involve iterative algorithms that take longer than FGSM [2]. For these reasons, in order to have a fast and effective method for testing and data augmentation, we chose FGSM.

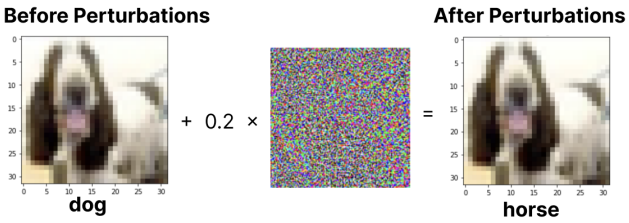

Figure 1. Visualized FGSM process showing predictions of an adversarial example from this paper implementation.

1.2. Convolutional Neural Networks (CNNs)

A Convolutional Neural Network (CNN) is a deep learning model that is particularly effective at processing and classifying images by taking advantage of convolutional filters. CNNs consist of multiple layers of filters that learn to extract meaningful features from the input images. The first layer of the network typically learns simple features such as edges and corners, while the subsequent layers learn increasingly complex features such as shapes and textures. Adversarial attacks such as FGSM have been researched heavily with CNNs making the architecture defined in **Section 6** an excellent choice for this paper [1, 2].

1.3. Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) are a type of neural network that are designed to handle sequential data by

processing input data one element at a time, while maintaining a hidden state that captures information about the previous elements in the sequence. An LSTM (Long Short-Term Memory) is a type of RNN that is particularly effective at handling sequential data, such as text or time series. While both RNNs and LSTMs are not typically used for image classification tasks, they could potentially be used in a computer vision problem involving sequential data, such as video classification.

For defending against adversarial examples, LSTMs could potentially be used in conjunction with CNNs to improve the robustness of the model. LSTMs could be used to model the temporal dependencies in the input data, which could help to identify and filter out adversarial perturbations that do not follow a coherent sequence. A hybrid model was chosen for this paper involving using a CNN-LSTM hybrid defined in **Section 6**. The beginning CNN hopes to mitigate some of the weaknesses of LSTMs with computer vision problems while still allowing for the benefits of LSTMs for handling temporary data for potential features for defending against adversarial examples.

1.4. Adversarial Training

Adversarial training is a technique used to improve the robustness of machine learning models against adversarial attacks. For the context of this paper, it involves the following pipeline:

1. Train a simple baseline model on the CIFAR-10 dataset.

2. Generate adversarial examples using FGSM. This involves computing the gradient of the loss function with respect to the input image, and then perturbing the image in the direction of the sign of the gradient. The magnitude of the perturbation is controlled by a hyperparameter called ϵ.

3. Add the adversarial examples to the dataset and train the model on the augmented dataset. This process is repeated for multiple epochs.

4. Evaluate the model on a test set that was attacked with varying ϵ to measure its performance against adversarial attacks.

Adversarial training with FGSM has been shown to improve the accuracy and robustness of computer vision models against adversarial attacks [3]. Particular for this paper, we incorporated the *torchvision* module in order to before transformations on images for data augmentation [4]. While this module is expansive for integrating with PyTorch, it does not have an implementation for FGSM transformations which is what we sought to implement ourselves.

2. Results

To create our adversarial examples, we implemented a custom *transformation* class called **FGSMTransform** and **FGSMTransformCNNLSTM** following the specifications of the *torchvision transformation* specifications [4]. They take in an image and a value ϵ, calculates the perturbation to add based on the gradients of a given baseline model, and returns the adversarial example. We then trained a two CNN & CNN-LSTM hybrid models for baselines. Using our transformation, we performed an FGSM attack on each image in the CIFAR-10 training set, and we augmented the train dataset by concatenating those results with the original train set. With each model, we varied the ϵ that transformed the training set. Finally, we tested the models with a new set of FGSM-attacked images with varying degrees of testing ϵ. The resulting accuracies and losses are summarized in **Table I-II** and **Figures 4-7** in **Section 5**.

To be noted here is that we first trained multiple models of the CNN architecture to tune the hyperparameter of ϵ before training the CNN-LSTM models with the chosen parameters, since the latter takes significantly more time to train. With the CNN model, we tested 8 ϵ values ranging from 0.005 to 0.5, and we found that as ϵ increases, the improvement seen in the model slows down exponentially. We chose the 4 ϵ that resulted in the most significant improvements for our final test.

Before encountering any adversarial perturbations, the CNN and CNN-LSTM hybrid architectures had accuracies of 80.374% and 79.001%, respectively. However, when subjected to adversarial examples with an epsilon value of 0.1, their accuracies dropped drastically to 9.375% and 12.679%. On the other hand, our top-performing robust models effectively defended against these attacks and achieved validation accuracies of 75.488% and 74.056% for the CNN and CNN-LSTM hybrid architectures, respectively. These values represent significant improvements of 64.142% and 61.375% over the baseline models.

We observed that generally, greater ϵ lead to greater resilience against FGSM attacks. For the CNN, the model trained on $\epsilon = 0.5$ observed a decrease in both accuracy and loss from $\epsilon = 0.2$ as shown in **Figures 4 & 5**. As such, we conclude that $\epsilon = 0.2$ is the optimal value.

3. Discussion

The above experiments revealed several key insights about adversarial training for greater resilience to adversarial examples.

There became a clear improvement to an adversarially trained model's resistance to adversarial examples with higher values of training ϵ. In fact for $\epsilon = 0.2$ for both the CNN and CNN-LSTM architectures, there was an observed increase in the performance of the model over adver-

sarial examples. One can observe that by the $\epsilon = 0.5$ model trained on the CNN that this is not a completely linear correlation. ϵ being a hyperparameter of adversarial training demonstrates that it is possible for the model to "overfit" to adversarial examples in the training set leading to a decrease in accuracy across adversarial examples. For this reason, we were able to conclude that amount the tested values of training ϵ, 0.2 was the optimal hyperparameter for both architectures.

Another consideration involves the conditions of training. For all models, we limited training to a maximum of 10 epochs due to the intensive time needed to perform both adversarial data augmentation and training for each model on a CPU. For training all five training epsilons on each architecture, it took ≈ 3.5 hours for the CNN architecture and ≈ 7 hours for the CNN-LSTM hybrid. While tuning the base architectures, we observe that it became more clear on the performance on adversarial training when the baseline model had a higher initial accuracy, the adversarially trained models performed better as well due to fewer false positives for classification from adversarial attacks that were caused by initial generalization issues from the baseline model. If greater resources such as access to a GPU were possible for this paper, then some improved benchmarks for the baseline model could be possible and in conjunction, the adversarially trained models could potentially see greater performance.

With the comparison between CNN and RNN architectures, we can reference **Figures 2 & 3**. We can observe that in **Figure 2**, for higher values of testing ϵ, the CNN-LSTM baseline architecture is able to slightly outperform in accuracy the CNN architecture despite having a slightly weaker baseline accuracy. However, the strength of CNNs becomes clear for the optimal training ϵ of 0.2. For these architecture, as shown by the differences between baseline and adversarially trained accuraccies, the performance increases becomes more clear for higher values of testing ϵ as CNNs have a slightly higher accuracy for these as well as a higher change in accuracies. This may verify some of the weaknesses of LSTMs with handling adversarial trained data. Adversarial peturbations are not generated sequentially which is the particular type of data that RNN and LSTMs are particularly strong at interpreting. This highlights the strengths of CNNs to handle spatial locality of data which is prominent for computer vision tasks allowing to not only have a better baseline performance but also adversarial performance.

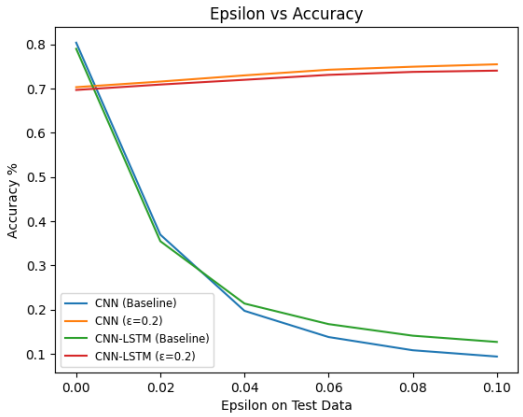

Figure 2. Testing accuracies for baseline and adversarially trained networks over testing ϵ.

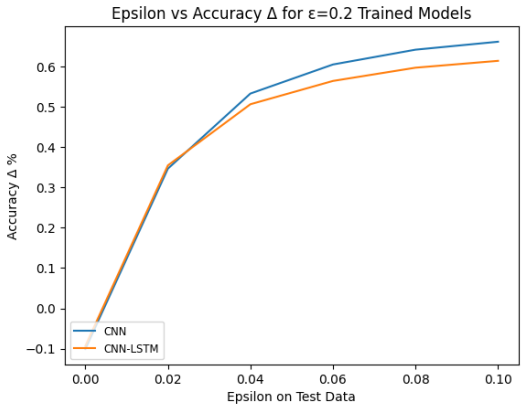

Figure 3. Changes in testing accuracies for baseline and adversarially trained networks over testing ϵ.

References

[1] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. 1

[2] Alexey Kurakin, Ian J. Goodfellow, Samy Bengio, Yinpeng Dong, Fangzhou Liao, Ming Liang, Tianyu Pang, Jun Zhu, Xiaolin Hu, Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, Alan Loddon Yuille, Sangxia Huang, Yao Zhao, Yuzhe Zhao, Zhonglin Han, Junjiajia Long, Yerkebulan Berdibekov, Takuya Akiba, Seiya Tokui, and Motoki Abe. Adversarial attacks and defences competition. *ArXiv*, abs/1804.00097, 2018. 1

[3] Mst. Tasnim Pervin, Linmi Tao, Aminul Huq, Zuoxiang He, and Li Huo. Adversarial attack driven data augmentation for accurate and robust medical image segmentation, 2021. 2

[4] PyTorch. Torchvision. https://pytorch.org/vision/stable/index.html. 2

4. CNN Performance

	Accuracy (%) on Testing ϵ					
Trained ϵ	0	0.02	0.04	0.06	0.08	0.1
Baseline	80.4	36.9	19.7	13.8	10.8	9.4
$\epsilon = 0.005$	76.0	69.5	58.7	46.5	38.0	31.0
$\epsilon = 0.01$	75.4	72.7	67.5	59.8	51.0	42.9
$\epsilon = 0.015$	75.1	74.2	69.8	61.9	51.6	42.5
$\epsilon = 0.2$	70.3	71.6	73.0	74.3	75.0	75.5

Table 1. Accuracy of adversarially trained CNNs.

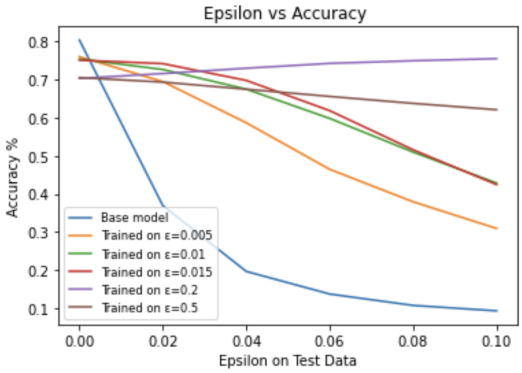

Figure 4. Accuracy of adversarially trained CNNs.

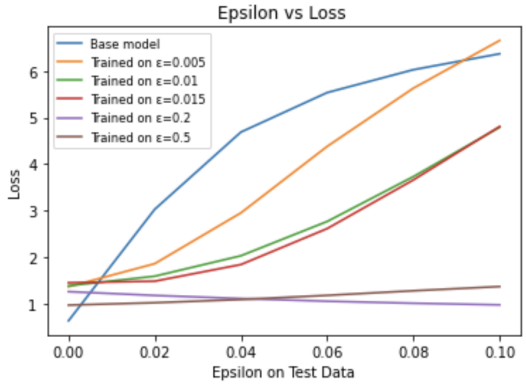

Figure 5. Loss of adversarially trained CNNs.

5. CNN-LSTM Performance

	Loss on Testing ϵ					
Trained ϵ	0	0.02	0.04	0.06	0.08	0.1
Baseline	79.0	35.5	21.4	16.7	14.1	12.7
$\epsilon = 0.005$	75.1	69.0	58.1	46.1	36.7	30.1
$\epsilon = 0.01$	73.3	70.1	62.9	52.3	42.7	35.6
$\epsilon = 0.015$	74.0	72.2	67.5	59.1	49.3	41.7
$\epsilon = 0.2$	69.7	70.1	72.0	73.1	73.8	74.1

Table 2. Loss of adversarially trained CNNs.

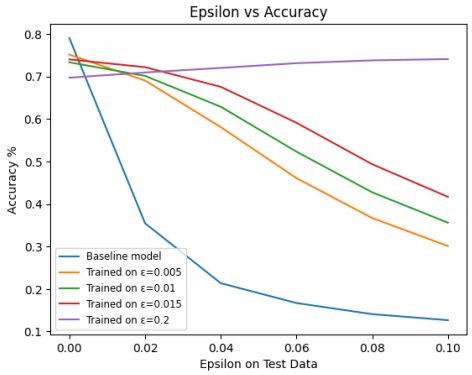

Figure 6. Accuracy of adversarially trained CNN.

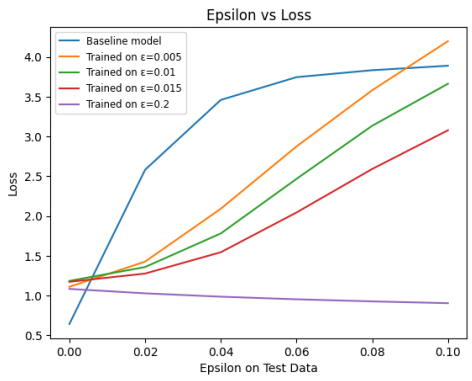

Figure 7. Loss of adversarially trained CNN-LSTM hybrids.

6. Architectures

Below is a description of all architectures and attacks implemented in this paper. For access to all of the code and models trained for this paper, please visit https://github.com/bliutech/adversarial-networks. Some further details on hyperparameter tuning are included in the files relevant to each architecture.

6.1. Attack Implementations

The adversarial attack was implemented in two stages: one FGSM function that generates an adversarial attack given the image, gradients, and perturbation ϵ; and one transformer that is compatible with Pytorch's torch.transforms module. For compatibility reasons, we implemented a custom CIFAR-10 data loader class that allows the transformer to calculate the loss and gradients of the image. During the data augmentation stage, we integrated the FGSM transformer with the data loader to create an augmented dataset consisting of the original CIFAR-10 images and their adversarial counterparts. During the testing stage, we varied the transformer and computed the validation accuracy of each model given a set of adversarial examples.

6.2. CNN Architectures

The CNN architecture takes an input tensor with three channels and applies a series of convolutional layers followed by ReLU activation functions and batch normalization layers. The network starts with a 3x3 convolutional layer with 32 output channels and a stride of 1, followed by ReLU activation and batch normalization. Then, there is another 3x3 convolutional layer with 32 output channels and a stride of 2, followed by ReLU activation and batch normalization. This pattern repeats with two more pairs of convolutional layers with 64 output channels, and then a final pair of convolutional layers with 128 output channels. The last convolutional layer is followed by a flattening layer that converts the output to a one-dimensional tensor, which is then passed through a fully connected layer with 64 neurons. This architecture also was used with a criterion of cross entropy loss, an optimizer of Adam and a learning rate of 0.001.

6.3. RNN Architectures

For the CNN-LSTM hybrid architecture chosen, the CNN component is similar to the architecture described in the **CNN Architecture** section, taking an input tensor with three channels and applying a series of convolutional layers followed by ReLU activation functions and batch normalization layers. After the CNN, the output is flattened and fed into an LSTM layer with 64 hidden units and 3 layers. The LSTM takes a sequence of inputs, so the output of the CNN is reshaped to have a time dimension before being fed into the LSTM layer. The LSTM layer is designed to capture temporal dependencies in the input sequence and generate a fixed-size representation of the sequence. Finally, the output of the LSTM layer is passed through a fully connected layer with 10 output units, which represents the classification output. The forward method of the model takes an input tensor and applies the CNN, followed by reshaping and feeding into the LSTM. The final output is generated by passing the output of the last timestep of the LSTM through the fully connected layer. This architecture also was used with a criterion of cross entropy loss, an optimizer of Adam and a learning rate of 0.001.