Convolutional Neural Networks and Adversarial Examples

Ari van Everdingen (V00988442)

University of Victoria

Abstract. Convolutional neural networks (CNNs) have proven themselves to be capable of classifying images with high accuracy. However, they are vulnerable to adversarial examples, which are inputs crafted specifically to cause the model to misclassify them. This project explores training CNNs with various techniques to become resistant to these adversarial examples. The examples are generated with Fast Gradient Sign Method (FGSM), and the techniques used to create more robust models are Adversarial Training, and Defensive Distillation. The Defensive Distillation method is analyzed, and a variation is used. An initial baseline CNN evaluated on a dataset generated with FGSM scores an accuracy of 20.4%. The final model, which is trained with Adversarial Training and Defensive Distillation variation, adds some noise to inputs as a preprocessing step and scores an accuracy of 59.8%.

1 Introduction

A Convolutional Neural Network (CNN) is a type of neural architecture designed for high dimensional images organized in arrays with some spatial quality, such as images. Their architecture involves convolutional layers and pooling layers followed by a feed forward network. Convolutional layers convolve learned kernels (small tensors) over inputs to create feature maps which indicate where the feature the kernel is looking for appears. Ideally, early convolutional layers identify simple features, such as edges or corners, and later layers identify more complex features, such as faces or body structures. Pooling layers take averages over input feature maps in order to reduce dimensionality, and improve spatial invariance. Finally, typical of classifiers, the output layer has a softmax activation to convert the model's output to a distribution over the set of classes [1]. This type of architecture has proven to be very effective in image classification tasks. However, CNNs are vulnerable to adversarial examples. Adversarial examples are inputs that have been perturbed specifically to cause misclassification. These perturbations should be subtle so that the resulting adversarial example still obviously belongs to the original class (see Fig. 1). If these models are deployed in critical situations, misclassification may have significant consequences such as self-driving car crashes. Therefore, creating models that are robust to such tampering is an important prerequisite to their deployment. This project seeks to create a safer model by being able to accurately classify adversarial

examples. The primary metric examined in this project is accuracy. If the index of the maximum value of the model's output (argmax) matches the correct class, this is considered a correctly classified example. Accuracy is used because a reduced accuracy is the most obvious impact of adversarial examples.

Fig. 1. The model confidently classifies the source image as a frog. When an adversarial perturbation is added to the source image (multiplied by a small value ϵ), the model somewhat confidently misclassifies the image as a cat.

2 Related Work

Adversarial training has been shown to be effective. On the MNIST dataset (black and white images of hand drawn digits 0 through 9[2]), adversarial training significantly decreased error rate on a dataset composed of adversarial examples, from 89.4% to 17.9% [3]. Defensive Distillation has been shown to reduce the success rate of adversarial examples from 95.89% to 0.45% on the MNIST dataset, and from 87.89% to 5.11% on the CIFAR-10 dataset (described in next section) [4]. Note that the metrics used in these papers are referenced (error rate and success rate), as opposed to accuracy as used in this project. It is unclear if the adversarial examples are made from images including those the models originally classified incorrectly, like this project does. This potential mismatch makes it hard to directly compare results.

3 CIFAR-10 Dataset

The CIFAR-10 dataset is a collection of 60,000 32x32 colour images, divided evenly into ten classes (See Fig. 2). There is a total of 50,000 training images, and 10,000 test images. Each pixel in an image has channels for red, green and blue, so the dimensions of each image is 32x32x3 [5].

The preprocessing steps taken were very simple: all values were divided by 255 to normalize pixel values between 0.0 and 1.0 (Min-Max normalization). Then, labels were encoded as one-hot vectors.

Fig. 2. An example from each of the 10 classes in the CIFAR-10 dataset

4 Fast Gradient Sign Method

The method used to generate adversarial examples is called Fast Gradient Sign Method (FGSM) [3]. FGSM seeks to do the opposite of Gradient Descent: perturb the input in the direction that increases loss the most.

$$x_{adv} = x + \epsilon \cdot sign(\nabla_x J(\theta, x, y_{true})) \tag{1}$$

where x_{adv} is an adversarial example, ϵ is a tunable parameter (in this project unless otherwise stated $\epsilon = 0.01$), $sign()$ takes the sign ($+1$ or -1) of each element in its input, $J(\theta, x, y)$ is a loss function, θ is the model's parameters, x is the input, and y_{true} is the true label for x (See fig. 3 for examples of FGSM applied with various ϵ). In order to test a model's robustness against adversarial examples, *perturbed datasets* were created. These datasets were composed of adversarial examples using FGSM with the CIFAR-10 test set as the source. The source images were selected regardless of whether the model originally classified them correctly or not. This is important to consider when comparing results to other works that may have only used images that were originally correctly classified. Note that the gradients used in FGSM must be from the tested model, so a unique perturbed dataset is created to test each model.

5 Baseline Model

The goal of this baseline model was to have good performance on the original dataset without overfitting so that we get a good idea of the impact of adversarial examples without a poor underlying model hindering understanding. A summary of the model architecture is given in Table 1).

The regularization techniques used are Batch Normalization layers, Dropout layers, and data augmentation. Batch Normalization seeks to reduce internal covariate shift: during training, the distribution of the inputs to each layer changes

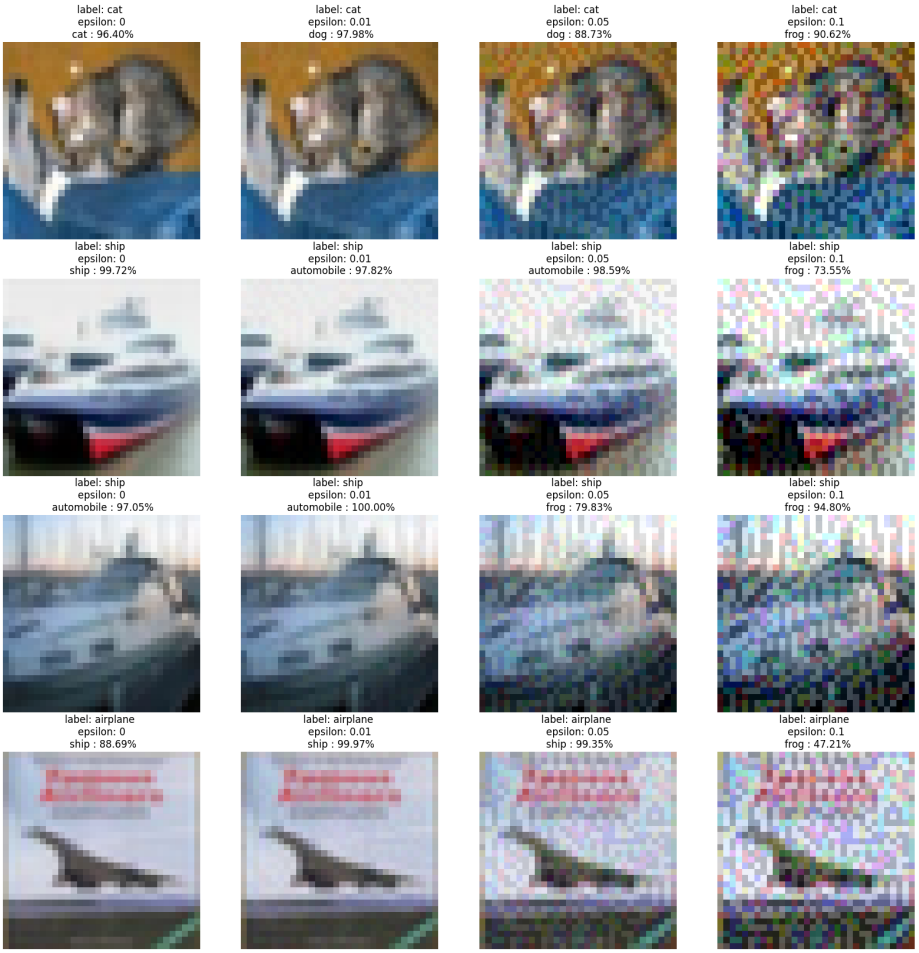

Fig. 3. Adversarial examples of 4 different images with varying ϵ values, and the baseline model's corresponding predictions. Note that since the model does not have perfect accuracy, it sometimes gets the image wrong, despite no perturbation having been applied.

Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 64)	36,928
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 64)	256
conv2d_5 (Conv2D)	(None, 8, 8, 64)	36,928
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 64)	0
dropout_2 (Dropout)	(None, 4, 4, 64)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 10)	10,250

Table 1. Summary of the model layers as output by Tensorflow. There are 150,954 total parameters. After [6]

each time parameters are updated. To counter this, Batch Normalization normalizes layer inputs. This speeds up training, as layers don't have to become accustomed to new input distributions each batch, and also provides some regulization [7]. It is common knowledge that normalizing inputs can help models learn - this method simply normalizes the inputs of each layer. The next regularization technique is the Dropout method. Dropout layers deactivate certain neurons and their connections in a given layer at train time. At test time, all neurons are active. This idea mimics training many models and then combining them at test time to improve generalization without actually having to train many models. This method is thought to encourage the model to recognize flexible features that work well with a random collection of other features [8]. Finally, data augmentation was used, applying random horizontal flips, adjusting brightness and contrast, and shifting the image horizontally and vertically. Categorical cross-entropy was used as the loss function. The architecture described here is used for all other models in this project. Any differences will be explicitly stated.

The baseline model has an accuracy of 0.828 on the CIFAR-10 test set, and 0.204 on its perturbed dataset. This is a significant decrease in accuracy, but is this decrease actually from FGSM? Perhaps the baseline model simply doesn't handle noise or perturbations very well. To test this, a new dataset was created with random adversarial examples. These new examples were created as follows.

$$x_{adv} = x + \epsilon \cdot R \tag{2}$$

Where R is a tensor in the same shape as x of 1s and -1s picked uniformly at random. Note that this R term replaces the $sign()$ term in Equation 1. On this new dataset, the baseline model scores an accuracy of 0.830. This illustrates how even though the baseline model performs well on noisy data, FGSM exploits the model's gradients to create examples it will still misclassify.

6 Adversarial Training

The first technique used to create a more robust model is called Adversarial Training. It consists of augmenting the training set with adversarial examples [3]. However, this is not completely trivial because a model's parameters frequently change throughout training. Therefore, one would have to create a new set of adversarial examples each batch to have examples available that could exploit the current model. Since limited compute is available, the following two methods were used:

1. Random Adversarial Training: Create a set of random adversarial examples before the start of training as an approximation. Computationally, this is much cheaper. These examples were created as described by Equation 2
2. Per Epoch Adversarial Training: Generate a set of adversarial examples every epoch. This way, hopefully, a new set of examples is generated before the old set is completely obsolete.

Using these two methods to augment the training dataset, we get two new corresponding models. Both had identical architecture and hyperparameters as the baseline model. Their performances are described in Table 2

	CIFAR-10	Perturbed Dataset
Random Adversarial Training	0.830	0.446
Per Epoch Adversarial Training	0.818	0.388

Table 2. Accuracies of the two adversarially trained models against the CIFAR-10 test set, and their respective perturbed datasets

Both models have similar performance on the CIFAR-10 compared to the baseline model, but do about twice as well on their perturbed datasets. It's unintuitive that the Random Adversarial Training model accurately classified 7% more of its perturbed dataset than the Per Epoch model. It's possible that frequently changing the test data introduces variance to the training that hurts performance. This could be tested by training a model with random adversarial

examples, but generating new random adversarial examples every epoch. Alternatively, the difference could also just be up to noise in training and initializations, or something else entirely. Unfortunately, determining the true reason for this difference is outside of the scope of this project.

The improved performance from Adversarial Training is also reflected in the magnitude of the gradients used in FGSM. Even though the magnitude does not affect the actual perturbation, it describes how much the loss will respond to the perturbation. A larger gradient means FGSM will increase the loss more, therefore making misclassification more likely. The gradient norms of the baseline model and the two Adversarial Training models are presented in Fig. 4

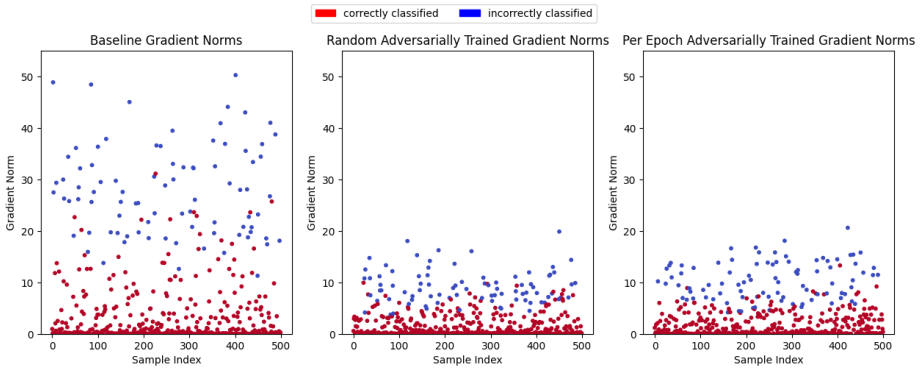

Fig. 4. These plots show the gradient norms of the baseline model, and the two Adversarial Training models. The gradients used for these plots are exactly the gradients used in FGSM (Equation 1). The euclidean norm was used despite the gradients' high dimensionality, as we are only interested in the magnitude of the gradients, not similarities between gradients. Also note that we care the most about the gradients associated with examples that our model initially correctly classifies (coloured in red). This is because if an example that is misclassified from the start is perturbed, even though the associated loss may increase and the classification may change, it can't get "more" misclassified (we are only considering accuracy, so we are not considering if one misclassification is more or less wrong than another). This is perhaps slightly naive, as it would be better for example, if a dog were to be misclassified as a cat as opposed to a plane, but it simplifies analysis significantly. After [9]

7 Defensive Distillation

7.1 Defensive Distillation and its Flaws

Before describing Defensive Distillation, consider the softmax function with a temperature parameter:

$$softmax(z)_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{3}$$

Where the logits, z_i are the output of the model right before the softmax activation. The temperature parameter controls how smooth the resulting distribution is. As T approaches infinity, the distribution becomes uniform. As T approaches 0, the distribution becomes completely concentrated at the index of the logit(s) with the greatest value.

Defensive Distillation consists of the following training steps:

1. Train a teacher model with a high-temperature softmax on the original dataset. Note that the labels are in the form of one-hot vectors, or *hard labels*.
2. Replace the labels in the dataset with the corresponding outputs of the teacher model. These outputs are not one-hot vectors, and therefore are called *soft labels*.
3. Train a student model with the same softmax temperature and architecture on this new dataset. This student model is also referred to as the *distilled model*.
4. At test time, reduce the student model's temperature to $T = 1$.

The idea behind Defensive Distillation is the soft labels carry more information than the hard labels. A one-hot vector simply describes what class an image belongs to. A soft label will describe what class an image most likely belongs to, but also classes that are similar. For example, a hard label may say, "This is a cat", while a soft label may say "This is probably a cat, but also looks kind of like a dog". If our student model learns the connections between classes through soft labels, the result should be a model with better representations of each of the classes, and smoother decision boundaries (and smaller gradients) [4]. We can expect longer training times for the student because of the soft-labels. Computing the loss function, categorical cross-entropy, cannot be simplified as it is when the labels are one-hot vectors (matrix multiplication with a one-hot vector is equivalent to selecting an index of the matrix). Using Defensive Distillation, we get three resulting models: the teacher, the student (the distilled model), and the student after decreasing the softmax temperature. Their performances are described in Table 3.

These results are very interesting. First note that the two versions of the student models have identical accuracy on the CIFAR-10 test set. This is because changing the temperature does not change the classification of a given input. That is to say, the temperature does not affect which index of the output vector has the greatest value (and is therefore chosen as the classification). Second, there is negligible improvement from the Teacher (T=50) model, to the Student (T=50) model. If our student really did learn better representations of the classes and smoother decision boundaries, there should be an improvement at this stage. Third, the Student (T=1) model shows suspicious improvements on the perturbed dataset.

	CIFAR-10	Perturbed Dataset
Teacher (T=50)	0.858	0.374
Student (T=50)	0.812	0.376
Student (T=1)	0.812	0.798

Table 3. Accuracies of the three Defensive Distillation models against the CIFAR-10 test set, and their respective perturbed datasets. Note that the teacher model has decent performance on the perturbed dataset because it was trained adversarially. The Student (T=1) model has a surprisingly high accuracy on its perturbed dataset.

It does not seem that learning from the soft labels provided by the teacher benefited the student model at all. To see why, let's look at a random output of the teacher for an example used in training (as this example will also be used to train the student):

$$1.6e{-}8, 7.6e{-}7, 8.3e{-}4, 5.0e{-}3, 1.4e{-}4, 4.7e{-}4, 9.9e{-}1, 1.0e{-}5, 1.3e{-}6, 5.3e{-}8$$

This output is very concentrated on the sixth index, which has a value of about 0.99, leaving the other 9 indices to fight over the remaining 0.01. Despite the teacher model having a high softmax temperature, the output is still very concentrated. This is because the model is still pushed to learn the one-hot vectors that are the labels. This results in the model simply producing more exaggerated logits that still give a very concentrated distribution. It is hard to believe that the student model would be able to extract any useful information from these values. One could argue that at the very least, we can see which class the model believes to be the second most likely. In this example, the second greatest value is at the third index, with a value of about $5.0e-3$. However, it is likely, especially after looking at average loss values over an entire batch, that this simply looks like noise to the model. In addition, any information our student model does manage to extract, the teacher model already knows. Why are we training a new model with the exact same architecture to relearn this? Using this reasoning, the lack of improvement of the Student (T=50) model seems reasonable.

If the Student (T=50) model is a bust, where does the sudden robustness of the Student (T=1) model come from? Plotting the gradients of these models can offer some insight in where to investigate (See Fig. 5)

If the gradients of the Student (T=1) model are all 0, then the FGSM perturbation, which uses these gradients, will be empty! FGSM considers the sign of these gradients, and the sign of 0 is simply 0. Therefore, the seemingly remarkable performance of the Student (T=1) model comes from FGSM not doing anything. In fact, if we test this model on the perturbed dataset from the Student (T=50) model, the prodigal Student (T=1) model has an accuracy of 0.376, the exact same accuracy of the Student (T=50) model. This again comes from the temperature having no influence on the actual decision boundaries. Therefore, the only impact of setting the temperature back to T=1 is to hide the gradients. However, since we are using FGSM which requires perfect knowledge of

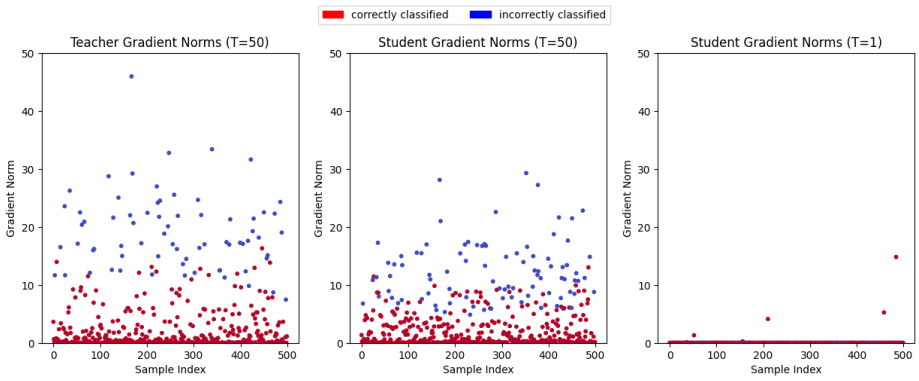

Fig. 5. Gradient norms of the three models from Defensive Distillation. The Student (T=50) shows slightly smaller gradients, but considering only gradients for correctly classified examples, the decrease is very little. Perhaps more interesting, is how the gradients almost all appear to be 0 for the Student (T=1) model. Note that in the rightmost plot, there were a few points with very large gradient magnitudes approaching 2,000. However, all of these points belonged to incorrectly classified examples (colored in blue), so were cut off to ease the comparison between the three plots.

the model's architecture and parameters to create adversarial examples, we are under the assumption that the adversary has white box access to the model. The adversary could simply increase the temperature to uncover the gradients, and therefore bypass any improvement Defensive Distillation offers.

The question remains, why are these gradients 0? Recall that decreasing the softmax temperature causes the resulting distribution to be more concentrated. In fact, if the temperature is decreased sufficiently, floating point precision is not precise enough to differentiate the resulting distribution from a constant distribution, or a one-hot vector. To illustrate this, If we take the same example used to create the teacher model's output above and input it to a version of the teacher model with softmax temperature T=1. The following one-hot vector is produced:

$$0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0$$

We get a 1 at the index of the greatest logit, and a 0 everywhere else. Assuming the model correctly classified this example, this is exactly the one-hot label for this example. Therefore the loss for this example is 0. Since the loss function is non-negative, this is a local minimum making the gradient at this point 0.

The idea of hiding gradients by decreasing the softmax temperature can also be applied to undistilled models. For example, if we take the Random Adversarial Training model, and decrease it's temperature to, say, T=0.01, many of the gradients become 0. This further demonstrates the ineffectiveness of the described distillation process.

In conclusion, Defensive Distillation has some flaws as follows:

1. The information encoded in the teacher's outputs is too small to trust the student model is learning from them as desired.
2. Any information the student learns from the teacher is already known by the teacher. Training a new model to relearn this information seems redundant.
3. Decreasing the student model's temperature at test time only hides the gradients.

7.2 A Variation of Defensive Distillation

In an attempt to fix the first flaw, the following method was devised:

1. Train a teacher model on the original dataset with softmax temperature T=1
2. After training, increase the teacher model's temperature to, say, T=4.
3. Train a student model with softmax temperature T=4 and the same architecture as the teacher with the soft labels output by the teacher model.

By increasing the softmax temperature of the teacher after training, we assure the outputs are smooth so the student can more easily extract information. This arguably also begins to tackles the second flaw, in that the information presented to the student model is slightly different to what the teacher model learned. Decreasing the student temperature at test time is obviously possible, but not necessary. Using this variation of Defensive Distillation, we get a resulting teacher and student model. Their performances are described in Table 4. The student scores with an accuracy almost 10% higher than the teacher. Once again, the performance is reflected in the plots of the gradients (See Fig. 6).

	CIFAR-10	Perturbed Dataset
Teacher (T=1)	0.869	0.351
Student (T=4)	0.821	0.448

Table 4. Accuracies of the two models from the variation of Defensive Distillation. Once again, the teacher was trained adversarially.

In conclusion, even though the original Defensive Distillation has flaws, a small tweak shows the promise of the method's idea.

8 Final Model and Conclusion

To create the final model of this project, the undistilled model with the best accuracy on its perturbed dataset was used as a teacher to produce one last student. The Random Adversarial Training model was selected, with an accuracy of 0.446 on its perturbed dataset. The resulting distilled student model scores an accuracy of 0.786 on the CIFAR-10 test set, and 0.488 on its perturbed dataset. A final trick that can improve accuracy is to add noise to inputs as a

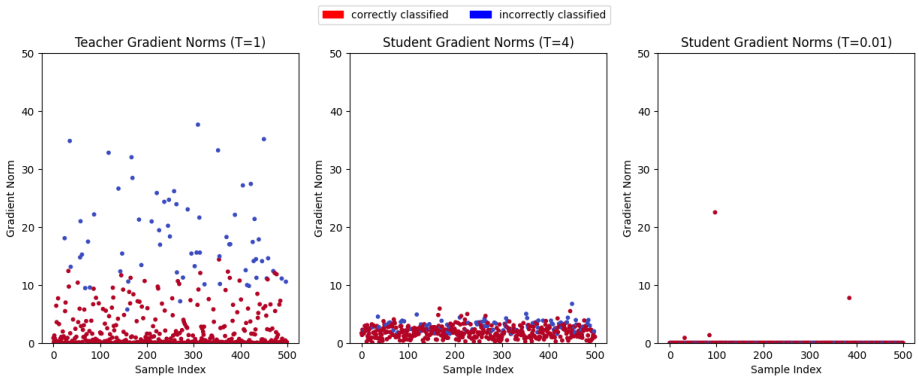

Fig. 6. Gradient norms of the models from the Defensive Distillation variation. This time, the Student (T=4) model shows dramatically reduced gradients. The low temperature model again shows 0 gradients, but this is uninteresting.

preprocessing step. Adding a random adversarial perturbation as described by Equation 2 to inputs results in an accuracy of 0.598 on its perturbed dataset. This is a significant improvement over the baseline model.

One unfortunate pattern that emerged during this project is how the Defensive Distillation variation seemed to result in decreased accuracy for the original CIFAR-10 dataset (See Table 4 and the accuracy of the final model described above). More investigation is required to see if this is caused by the technique, or can be avoided. Although the performance of the student model in the original Defensive Distillation also decreased, the original paper empirically claims the influence of distillation on accuracy is only moderate, decreasing by at most 1.37% on their models classifying CIFAR-10 [4].

In conclusion, the adversarial examples generated by FGSM exploit the gradients of a model. By using adversarial training, and a variation of Defensive Distillation, we reduce these gradients so that these perturbations have less influence on the resulting classification. The baseline model accurately classified 0.204 of its perturbed dataset, and the final model produced by this project accurately classified 0.598 of its perturbed dataset.

9 Limitations

This project has the following limitations. Evidenced by the vastly varying performance of Adversarial Training models, much experimentation was done with the exact datasets used for training in terms of the degree of augmentation used. Therefore, even if it was possible, not all models were trained with the exact same dataset. This inconsistency may have had unknown consequences. In addition, only one method of generating adversarial examples was used, using a specific ϵ value. Whether these models generalize to other methods, including black box methods, is not explored in this project. As stated in the conclusion, the distilled

student models produced by the variation of Defensive Distillation showed a decrease in performance on the original CIFAR-10 dataset. Whether this is directly caused by the distillation or is avoidable is not investigated in this project. Adversarial examples generalize across architectures surprisingly well [3]. This fact is also not explored at all, and could be considered when developing a pipeline for Adversarial Training. Adding noise as a preprocessing step is not analyzed or explored very thoroughly. In fact, when this trick is applied to the Random Adversarial Training model (not distilled), it scores an accuracy of 0.601 on its perturbed dataset. This is better than the final model performs, despite having a lower accuracy without the noise trick. Defensive Distillation was inspired by Distillation, in which the student is trained on the teacher's outputs, but has a simpler architecture [10]. Modifying the student's architecture relative to the teacher would be another interesting avenue to explore, in terms of variations of Defensive Distillation. Finally, Adversarial Training and Defensive Distillation are not the only techniques to create models resistant to adversarial examples. Other methods may have better performance, or work well in tandem with the methods explored in this project.

References

1. Y. Lecun; L. Bottou; Y. Bengio; P. Haffner: Gradient-based learning applied to document recognition, in Proceedings of the IEEE, November 1998
2. Y. LeCun and C. Cortes: The mnist database of handwritten digits, `https://yann.lecun.com/exdb/mnist/`, last accessed 2024/12/03
3. I. J. Goodfellow, J. Shlens, and C. Szegedy, Explaining and harnessing adversarial examples, in Proceedings of the 2015 International Conference on Learning Representations. Computational and Biological Learning Society, 2015.
4. N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami: Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks `https://doi.org/10.48550/arXiv.1511.04508`
5. A. Krizhevsky: CIFAR-10 and CIFAR-100 datasets, `https://www.cs.toronto.edu/~kriz/cifar.html`, last accessed 2024/12/03
6. A. Kumar: Achieving 90% accuracy in Object Recognition Task on CIFAR-10 Dataset with Keras: Convolutional Neural Networks https://appliedmachinelearning.wordpress.com/2018/03/24/achieving-90-accuracy-in-object-recognition-task-on-cifar-10-dataset-with-keras-convolutional-neural-networks/, last accessed 2024/12/03
7. S. Ioffe, C. Szegedy: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift `https://arxiv.org/pdf/1502.03167`
8. Haibing Wu, Xiaodong Gu: Towards Dropout Training for Convolutional Neural Networks, `https://doi.org/10.48550/arXiv.1502.03167`
9. S. Kulkarni: Pytorch implementation of Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks `https://github.com/shreyas269/Pytorch-defensive-distillation`, last accessed 2024-12-03
10. G. Hinton, O. Vinyals, J. Dean: Distilling the Knowledge in a Neural Network `https://doi.org/10.48550/arXiv.1503.02531`