AML Final Project Report

Discussion about Adversarial examples: Adversarial examples can be defined as inputs which are designed by adversary or attacker with the intent of making machine learning model to commit mistakes. Attacker can add controlled amount of noise to an input and network can be fooled into classifying erroneously.

Adversarial Example attacks on neural network are classified into two categories:

1. Targeted Adversarial Examples
2. Non - targeted Adversarial Examples

The only difference between targeted and non-targeted attacks is that targeted adversarial examples are generated with the intent of making machine learning model to classify that input example into targeted output class. However, the non-targeted examples are created with intent of making machine learning model to classify the input into wrong class but not any targeted class.

Paper "*Towards Deep Learning Models Resistant to Adversarial Attacks*" [1] mentions a lot of methods for generating adversarial examples. From methods mentioned in the research paper, we have implemented two methods for generating adversarial examples. These methods are as follows:

1. Fast Gradient Sign Method (FGSM).
2. Iterative Fast Gradient Sign Method (I-FGSM).

Paper "*Explaining and harnessing adversarial examples*" [2] discusses both of these attacks and their mathematical implementation in details. We have implemented both of these algorithms in tensorflow 2.0. We have used FGSM (Fast Gradient Sign Method) for generating untargeted adversarial examples while I-FGSM (Iterative Fast Gradient Sign Method) for generating targeted adversarial examples.

Fast Gradient Sign Method: Formula that describe the essence of FGMS is as follows:

$$x' = x + \epsilon \,.\, sign\left(\nabla_x J(\theta, x, y)\right)$$

where, $\nabla_x J$ is gradient of the loss function with respect to input x.

y is true label corresponding to input x.

This algorithm uses $sign$ function. This implies that when the gradient is positive, we need to increase the intensity of pixel, and if gradient is negative, we need to decrease the intensity of pixel. This attack leverages the linearities in the machine learning models such as neural network. The neural network tends to classify the adversarial input to wrong output class. ϵ is the factor the controls the amount by which intensity of each pixel is changed. During our analysis of FGSM, we have found out that as we increase the value of ϵ, adversarial examples are generated almost reliably.

Iterative Fast Gradient Sign Method: As indicated in paper "*Adversarial Examples: Attacks and Defenses for Deep Learning*" [1], the iterative methods yield more better results than one-shot methods. So, we have used I-FGSM for generating targeted adversarial examples. Following mathematical formulation provides the essence to I-FGSM.

$$x_t = x$$

$$x_{t+1} = x_t - \alpha \, . \; sign\left(\nabla_x J(\theta, x_t, y)\right)$$

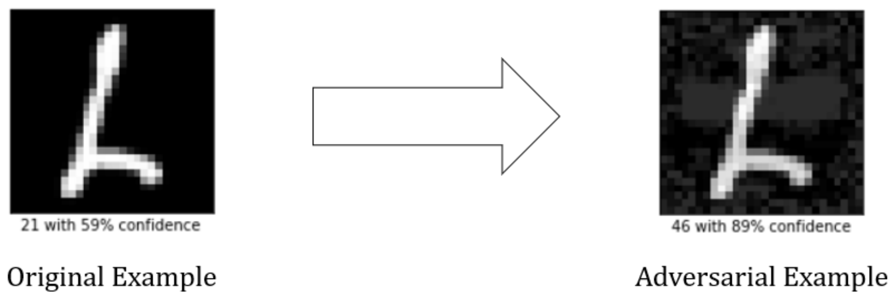
Where, $t \; = 0, 1, 3, \ldots \ldots \ldots$

x is input example.

$\alpha = \frac{\epsilon}{number\ of\ iterations}$

α in the equation controls the amount of change to the intensity of pixels. Iterative sign method takes number of epochs as input, which tells the method how many iterations to run before stopping and outputting the result.

Fig. 1 illustrates generation of an adversarial example from EMNIST data set, which we have generated during our project analysis. Figure on the left side is actual image taken from validation set of EMNIST data set. We are able to predict the example correctly with 59% confidence with our basic neural network. We generated adversarial image from this image, setting target class as 46, we are able to see that model predicts the adversarial image in class 46 with 89% confidence, however both images look similar.



Target Attack with target label 46

Fig 1. I-FGSM Adversarial Example

Discussion about Defensive Distillation Deep Neural Network: We discuss about Neural Network Distillation and its procedure from the paper "Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks" [3].

Distillation of neural networks is a training process to reduce the size of deep neural network (DNN) architectures in order to reduce its computational complexity so that deep learning could be deployed in resource constrained devices like smartphones. This process involves training a small DNN using the knowledge transferred from a larger DNN so that small DNN generalizes in the same way as the large model. This approach to distillation was introduced by Hinton et al [4]. In this paper [3], distillation is used in a way to provide security against adversarial examples. The knowledge is extracted from the first DNN trained in the form of class probability vectors known as soft target labels and they are used as input labels instead of hard labels to the second DNN. This shows that knowledge acquired by a DNN during training phase is not only present in the weight parameters but also in class probabilities produced by the output layer of DNN, that is, softmax layer.

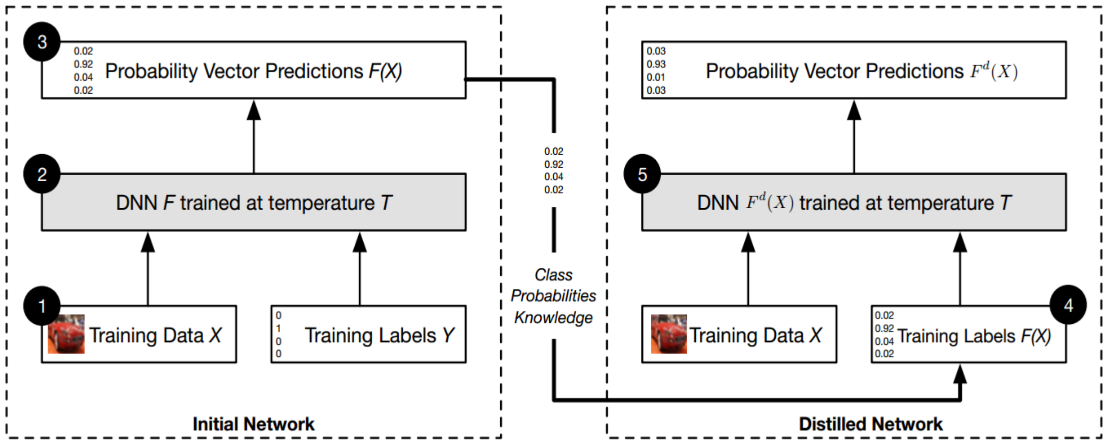


Fig. 2 Overview of Defensive Distillation

Given below are steps to perform defensive distillation also illustrated in Fig. 2 taken from the paper:

1. Input of the initial neural network is X which is a set of training samples and their class labels is Y(X) also known as hard label which is expressed in one-hot vector format.
2. Train a deep neural network F with a training set {(X, Y(X)): X ∈ X} and a softmax output layer at temperature T. Where, Z(X) is a logits vector produced by the last hidden layer which is normalized into F(X), a probability vector over the class of all possible labels.
3. Class probability vector $F_i(X)$ is the probability of input X to be in class i ∈ 0 to N − 1 according to model F with parameters θ_F expressed as

$$F_i(X) = \frac{e^{\frac{z_i(X)}{T}}}{\sum_{k=0}^{N-1} e^{\frac{z_k(X)}{T}}} \qquad \text{Equation 1}$$

where, $Z(X) = z_0(X),\ldots,z_{N-1}(X)$ are the N logits corresponding to the hidden layer outputs for each of the N classes in the dataset, and T is a parameter known as distillation temperature.

4. Form the new training set {(X, F(X)): X ∈ X} where, F(X) is also known as soft-target label and they contain additional knowledge about classes compared to a class label predicted by the neural network F.
5. This new training set is used to train another DNN with the same neural architecture as F with softmax output layer at same temperature T and this model is known as distilled model F^d.

This paper [3] also discusses about the impact of distillation on model sensitivity. For this discussion, an expression for model's sensitivity with respect to input variation is derived by taking first-order differential of Equation 1 with respect to model input parameter at temperature T:

$$\frac{\partial F_i(X)}{\partial X_j} = \frac{\partial}{\partial X_j}\left(\frac{e^{\frac{z_i(X)}{T}}}{\sum_{k=0}^{N-1} e^{\frac{z_k(X)}{T}}}\right)$$

Solving above equation results in final expression as given below:

$$\frac{\partial F_i(X)}{\partial X_j} = \frac{1}{T}\frac{e^{\frac{z_i(X)}{T}}}{\left(\sum_{k=0}^{N-1} e^{\frac{z_k(X)}{T}}\right)^2} \left(\sum_{k=0}^{N-1}\left(\frac{\partial z_i(X)}{\partial X_j} - \frac{\partial z_k(X)}{\partial X_j}\right)e^{\frac{z_k(X)}{T}}\right)$$

From the above equation, we can easily deduce that absolute value of adversarial gradient is inversely proportional to the distillation temperature for fixed values of $z_0(X),\ldots,z_{N-1}(X)$, that is,

$$\frac{\partial F_i(X)}{\partial X_j} \propto \frac{1}{T}$$

Therefore, high temperatures are required for defensive distillation during train time in order to reduce the model sensitivity with respect to small variations of input. Defensive distillation mechanism thwarts an effort by adversaries to evaluate the sensitivity of class change with respect to input feature and they further exploit this knowledge to find an effective adversarial perturbation δX. The adversaries attack using the fact that even small perturbations in the network can cause larger network output variations when adversarial gradient amplitude is high. This means that when adversarial gradient amplitude is small on distillation, estimating direction sensitivity becomes more complex hence larger amount of perturbation would be required for the same original input samples hence crafting adversarial samples is difficult on distilled networks. So, distillation generates smoother DNN classifier which is found to be more resistant to adversarial samples. Smoothing also helps the DNN to generalize well on samples outside of training dataset.

In order to do predictions on unseen inputs during test time, temperature is decreased back to 1. As increasing the temperature only affects the model's sensitivity and not the weight parameters and smaller sensitivity gained by using a high temperature is encoded in the weights during training and thus small sensitivity is also observed during test time.

Discussion about L_0, L_2 and L_∞ attacks: Paper [5] authors have developed three attacks for the L_0, L_2 and L_∞ distance metrics and have applied these attacks to the defensive distillation model and observed that distilled model fails against these attacks. Let's discuss about the three-distance metrics used for generating adversarial examples:

1. L_0 distance is a measure of total number of pixels that differ in their value between the original image x and altered image y.
2. L_2 distance is a measure of standard Euclidean root mean square distance between the original image x and altered image y.
3. L_∞ distance is a measure of maximum pixel difference between the original image x and altered image y.

In order to find an adversarial example for an image x, it is required to find adversarial perturbation δ which we can make to an image x that will change its classification from source class C(x) to target class t such that C(x) $\neq$ t using below problem:

$$minimize\ D(x, x + \delta)$$

$$such\ that\ C(x + \delta) = t$$

x + $\delta \in [0,1]^n$, so that the resultant image is still a valid image

where, D is either L_0, L_2 or L_∞ distance metric and x is fixed.

Above problem can also be formularized as

$$minimize\ D(x, x + \delta) + c.f(x + \delta) \qquad \text{Equation 1}$$

$$such\ that\ x + \delta \in [0,1]^n$$

where, f is an objective function such that C(x + δ) = t if and only if $f(x + \delta) <= 0$, c > 0 is a suitably chosen constant and D(x, x + δ) = $\|\delta\|_p$, $\|\delta\|_p$ is an L_p norm distance metric.

In order to ensure that x + δ ∈ $[0,1]^n$ condition is satisfied on generating adversarial example, we introduce a new modifier w so that we now optimize for w instead of δ for all pixels and equation for w in terms of δ for any pixel i of input sample is given as

$$\delta_i = \frac{1}{2}(tanh(w_i) + 1) - x_i \qquad \text{Equation 2}$$

Now, the three high-confidence adversarial attacks proposed in this paper are as discussed below:

1. L_2 Attack: In this attack, we search for an optimum value of w by using L_2 distance norm in below equation

   $$minimize\|\delta\|_2 + c.f(x + \|\delta\|_2)$$

   On substituting Equation 2 in above equation, we get

   $$minimize \left\|\frac{1}{2}(tanh(w) + 1) - x\right\|_2^2 + c.f(\frac{1}{2}(tanh(w) + 1))$$

with f defined as

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t - k)$$

where, x' is an adversarial instance of original image x and parameter k is adjusted to control the confidence with which the misclassification occurs.

2. L_0 Attack: L_0 distance metric is non-differentiable as a result it cannot be used for gradient descent calculation $g = \nabla(f(x'))$ at the adversarial instance. Therefore, an iterative algorithm is used and, in each iteration, L_2 attack is used to find pixels which are not important in generating adversarial examples. In each iteration, those pixels are identified that do not cause any change in the source class and forms a set of these pixels that need not be modified for further iterations as they are useless for creating adversarial samples. After each iteration, this set grows until by the process of elimination we get the minimal subset of pixels that can be altered to craft adversarial examples.

3. L_∞ Attack: Since L_∞ distance metric is non-differentiable, gradient descent cannot be computed. Therefore, below equation is optimized

   $$minimize\ \|\delta\|_\infty + c.f(x + \delta)$$

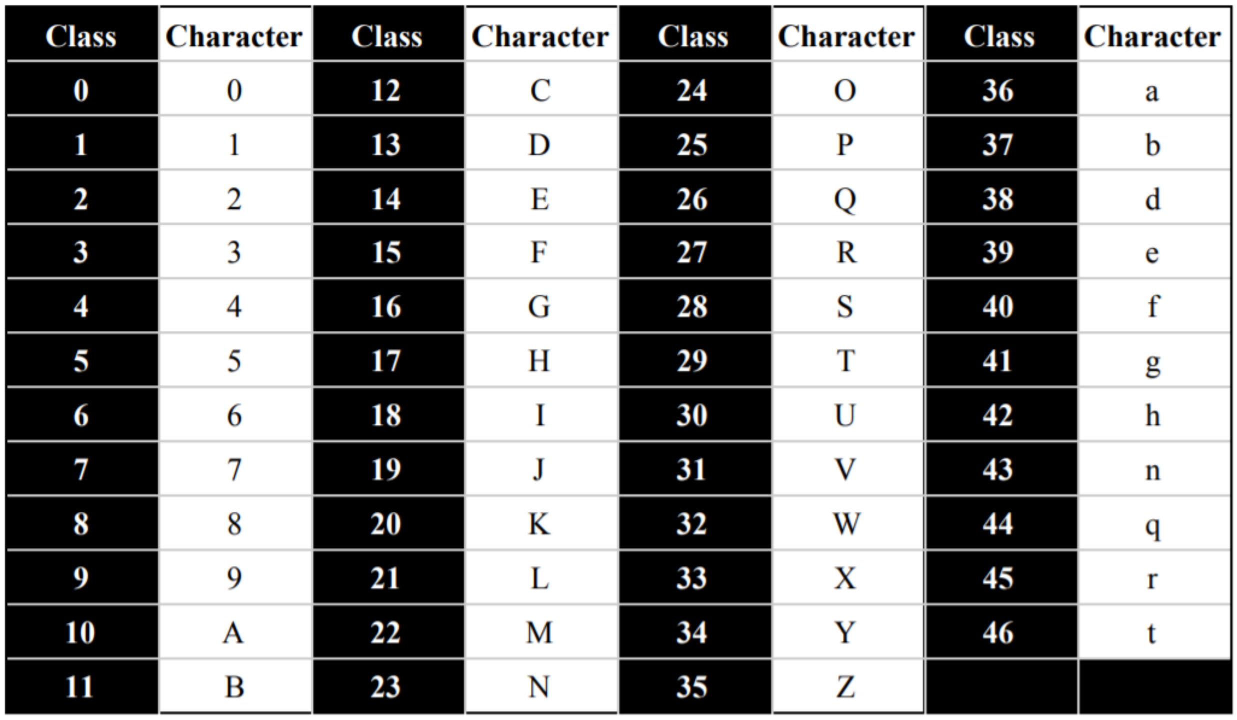
   However, it is found that gradient descent does not perform well using above equation.
   As gradient descent gets stuck oscillating between two suboptimal solutions. The solution to this issue is that an iterative algorithm is used and, in each iteration, below equation is optimized.

   $$minimize\ c.f(x + \delta) + \sum_i[(\delta_i - \tau)^+]$$

   where, initial value of parameter is τ is 1 and after each iteration, if $\delta_i < \tau$ for all i, τ is reduced by a factor of 0.9 otherwise iteration ends there. This solution prevents oscillation and value of c is taken very low and if L_∞ attack fails at a particular value of c then c is doubled until attack becomes successful. Algorithm is aborted if c exceeds a particular threshold value.

Evaluation and Observations

To evaluate our project, we have selected EMNIST balanced dataset that contains 47 balanced classes each which corresponds to unique handwritten letter or digit. Below table shows mapping of each character to a unique class. Each example in EMNIST data set is 28 x 28 example containing total of 784 pixels. Each pixel intensity ranges from 0 to 255. Before using the images, we have normalized the images to eliminate exploding gradients problem. Normalization ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network.



Class	Character	Class	Character	Class	Character	Class	Character
0	0	12	C	24	O	36	a
1	1	13	D	25	P	37	b
2	2	14	E	26	Q	38	d
3	3	15	F	27	R	39	e
4	4	16	G	28	S	40	f
5	5	17	H	29	T	41	g
6	6	18	I	30	U	42	h
7	7	19	J	31	V	43	n
8	8	20	K	32	W	44	q
9	9	21	L	33	X	45	r
10	A	22	M	34	Y	46	t
11	B	23	N	35	Z		

Code for evaluation can be found in $main.ipynb$ notebook. Some other supporting python scripts and models are as under:

1. adversarial_examples.py: It contains APIs for generating adversarial examples.
2. train_network.py: It contains APIs for creating and training basic and Distilled neural network.
3. plots.py: It contains APIs for plotting data in various forms. Plots computed during our analysis are attached int his report along with their thorough explanation.
4. li_attack.py: This script contains code from paper [5] author. This script contains L_∞ attack that we have reused and verified with new dataset.
5. l0_attack.py: This script contains code from paper [5] author. This script contains L_0 attack that we have reused and verified with new dataset.
6. l2_attack.py: This script contains code from paper [5] author. This script contains L_2 attack that we have reused and verified with new dataset.
7. models: This directory contains distilled models at different Distillation Temperatures.

Evaluation of Adversarial Examples: Fig. 3 illustrates first 20 examples from validation set of EMNIST dataset along with their labels. Using I-FSGM, we generated adversarial examples taking whole validation set as input and class 46 as target label. We have chosen iterations equal to 10 and value of ϵ equals to 0.01.

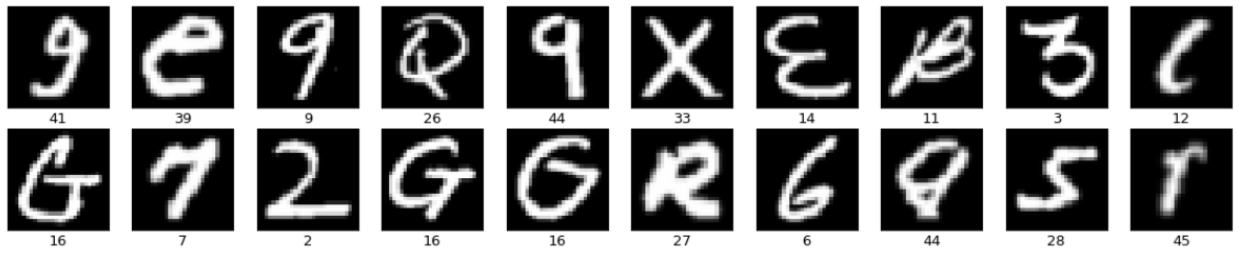


Fig. 3 First 20 examples from validation set.

Adversarial images generated using above mentioned parameters are illustrated in Fig. 4. We plotted only first 20 examples from dataset.

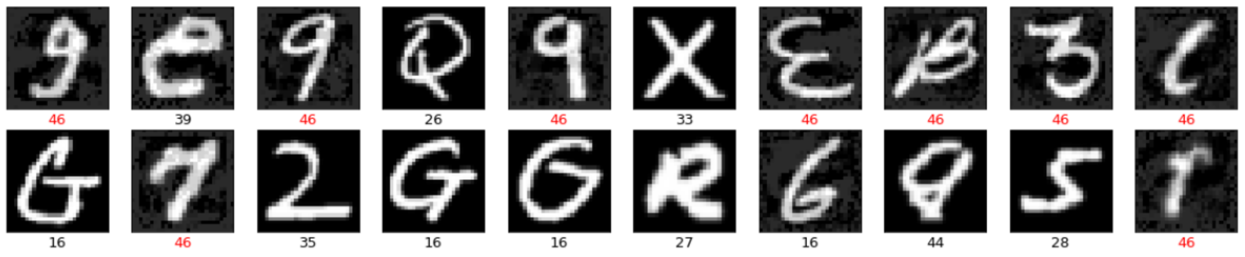


Fig. 4 First 20 examples from validation set after Adversarial attack.

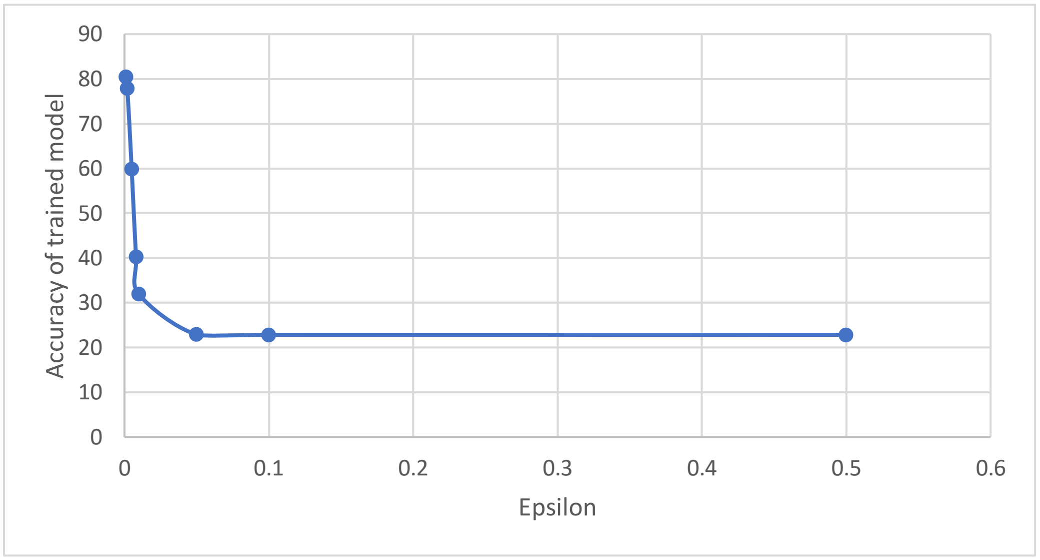
Above figure clearly indicates how addition of controlled amount of noise to original images actually fooled neural network to classify these images to class 46.

As paper [1] mentioned that effectiveness of adversarial attack actually depends upon the value of controlling parameter ϵ. As the we increase the value of ϵ, adversarial sample success rate should increase i.e. model should classify more and more adversarial examples into wrong class. Our analysis has confirmed this fact. Table 1 shows as we increase the value of ϵ model accuracy decrease as model is fooled by a greater number of adversarial examples.

ϵ	Accuracy of model with Adversarial Examples (%)
0.001	80.53
0.002	77.88
0.005	59.93
0.008	40.25
0.01	31.92
0.05	22.89
0.1	22.82
0.5	22.84

Table 1. Accuracy variation with ϵ

Graphical representation of Table 1 is as under:



Now, Let’s see how adversarial examples (generated from whole validation set) performed with our trained model. Fig. 5 clearly illustrates how targeted adversarial attack actually caused neural network model to predict images wrongly in class 46. In balanced EMIST dataset, each class have equal number of examples. But we can see that approximately 9500 images (out of total 18800 in validation set) are classified in class 46.

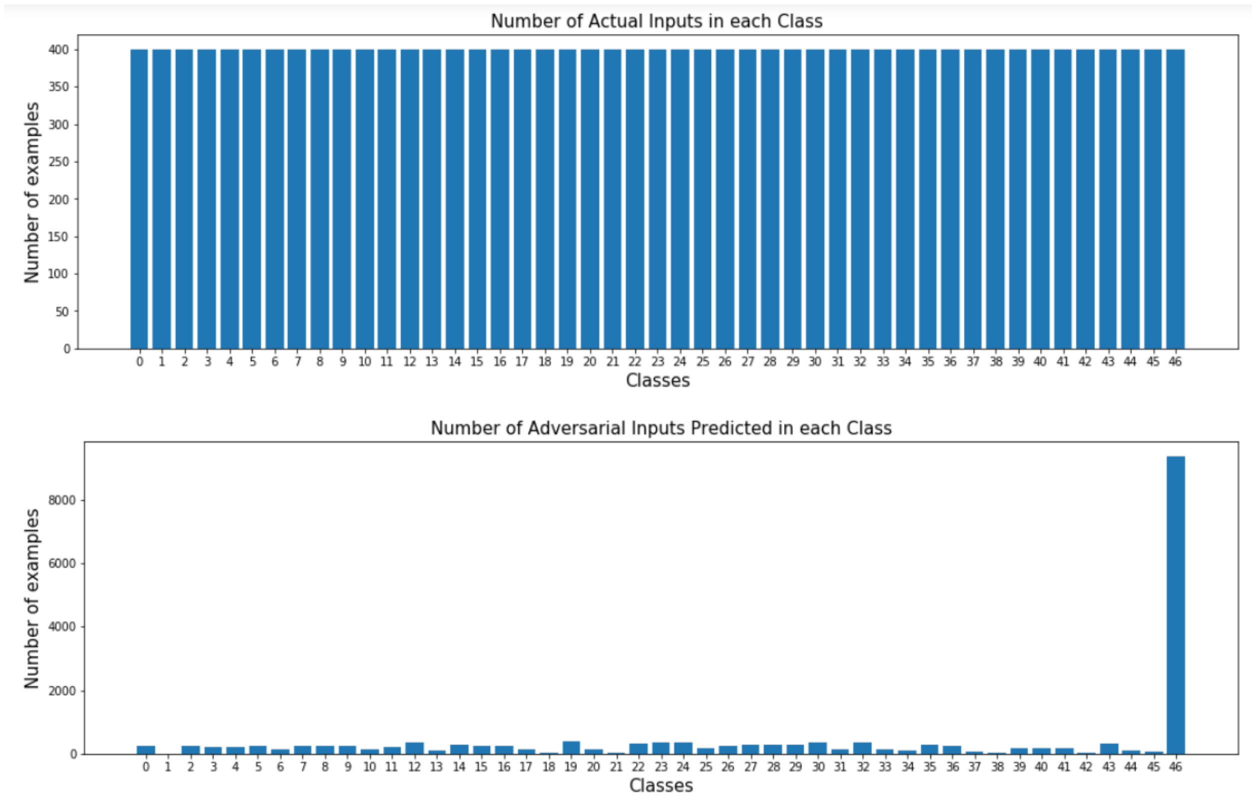


Fig. 5 Performance of Adversarial examples with trained model

Evaluation of Defensive Distillation DNN: As paper [3] mentions that Defensive Distillation neural network models provide security against adversarial examples and helps reducing their adverse effect on neural network. We have evaluated defensive distillation using the deep neural network architecture constructed for EMNIST data set. The architecture specific details and training parameters are given in the Table 2 and 3 respectively.

Layer Type	**EMNIST Architecture**
Relu Convolutional	32 filters (3 X 3)
Relu Convolutional	32 filters (3 X 3)
Max Pooling	2 X 2
Relu Convolutional	64 filters (3 X 3)
Relu Convolutional	64 filters (3 X 3)
Max Pooling	2 X 2
Relu Fully Connect	200 units
Relu Fully Connect	200 units
Softmax	47 units

Table 2. Distilled network architecture

Parameter	EMNIST Architecture
Learning Rate	0.01
Momentum	0.5
Decay Delay	-
Dropout Rate (Fully Connected Layers)	0.5
Batch Size	128
Epochs	50

Table 3. Training Parameters

We have conducted an experiment to measure the effect of temperature on adversarial sample success rate. The goal for this experiment is to identify the optimal training temperature which would result in maximum resilience towards adversarial samples for EMNIST based deep neural network. The adversarial success rate is measured for 1, 2, 5, 10, 35, 50, 100 set of distillation temperatures. Fig 6. Illustrates the plot for distillation temperatures vs percentage of source-target misclassification achieved by crafting adversarial samples. This graph indicates that as the distillation temperature increases, the success rate of adversarial examples decreases which implies that with increase in distillation temperature, model sensitivity to adversarial perturbations decreases which make the model resilient towards adversarial examples.

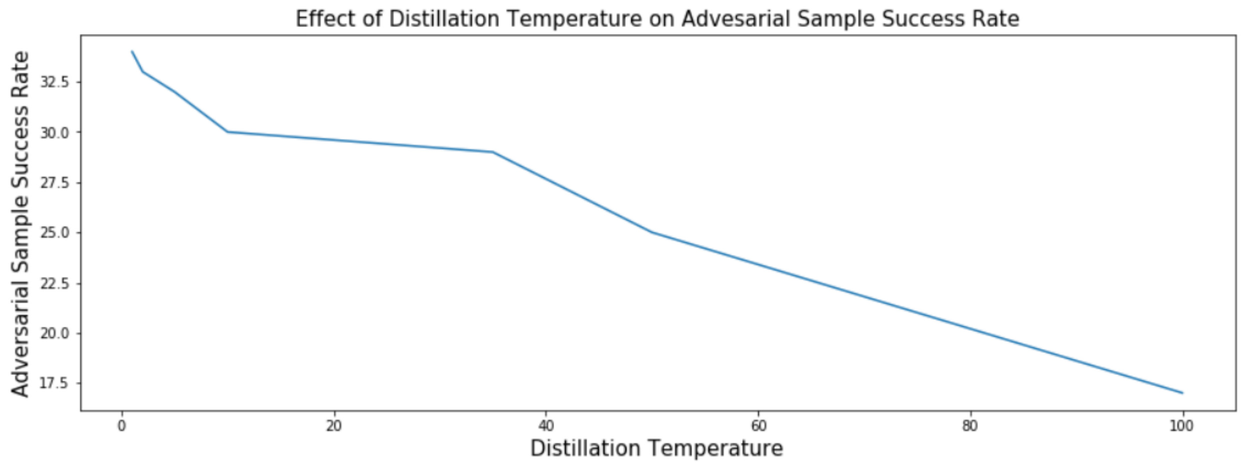


Fig. 6 Impact of Distillation Temperature on success rate of adversarial samples

As we see in Fig. 5 that basic neural network is fooled by the adversarial images causing it to classify majority of images in class 46. We used the same set of adversarial validation images and tested them with Defensive Distillation neural network. Architecture of Distilled network is defined in Table 2. We have trained this model at Distillation Temperature equals to100 with fifty epochs which is sufficient to achieve the accuracy comparable to our distilled model. Fig. 7 illustrates the prediction of distilled neural network when adversarial images are fed to it.

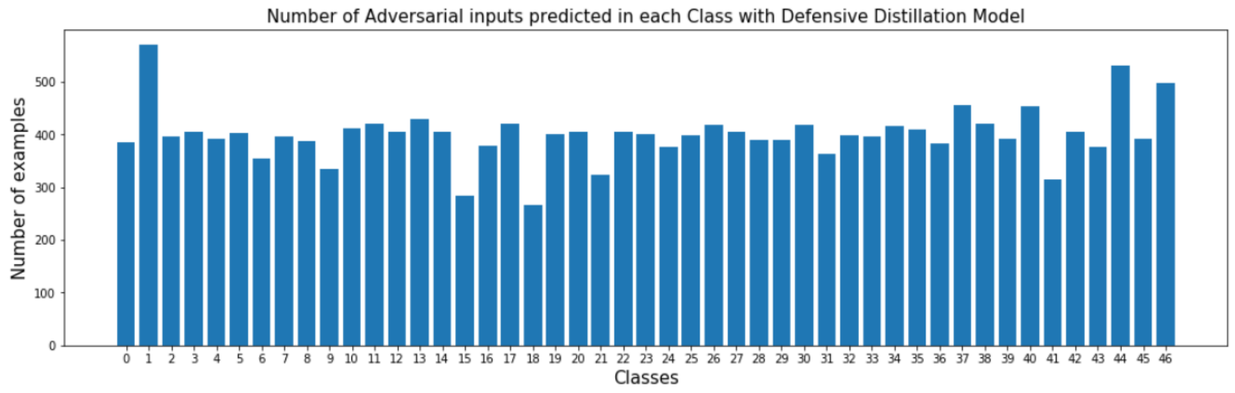


Fig. 6 Performance of Adversarial examples with Distilled model

This plot clearly shows that distilled model has filtered out a large number of adversarial examples in class 46. Fig. 7 shows that model accuracy of distilled model is around 86 % with adversarial images as input.

Note: We have already saved some of the Defensive Distillation models at different Distillation Temperatures and python notebook only loads these pre-saved models from *models* directory. This is because of the reason that each model took around 4 hours (on Google Collaboratory). So, it is better to load pre-saved model instead of training every time. The code to train the model can be found in train_networks.py (API *train_distillation*)

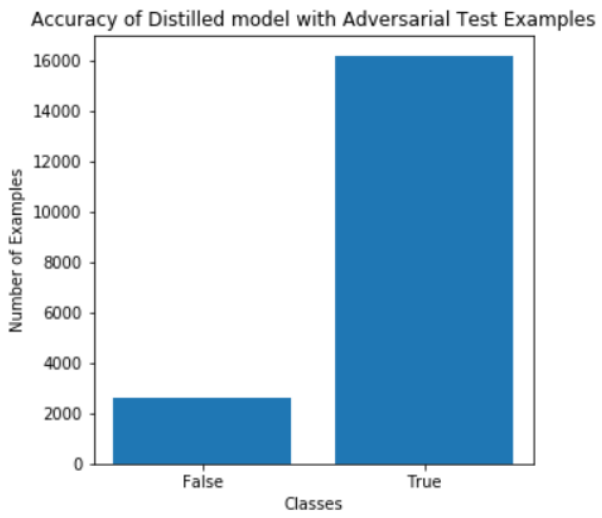


Fig. 7 Accuracy of Distilled Model with Adversarial Examples

We have conducted another experiment to compute the variation of classification accuracy between the EMNIST based model trained without distillation and with distillation over 1, 2, 5, 10, 35, 50, 100 set of temperatures.

As per our measurements, classification accuracy without distillation is 89.07%.

Given below is the plot of distillation temperature vs accuracy variation after distillation.

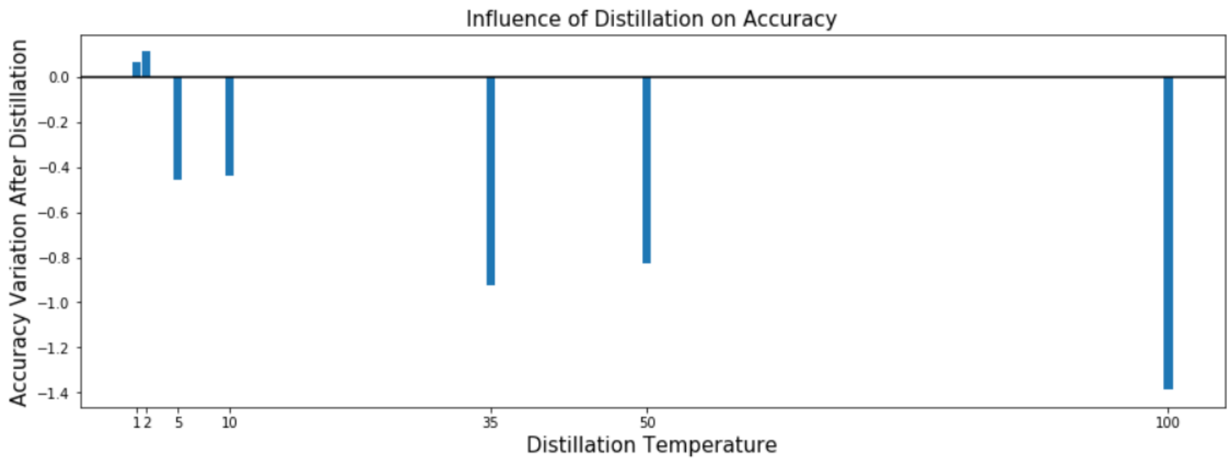


Fig. 8 Influence of Distillation on Accuracy

Paper [3] indicates that variations in accuracies introduced by distillation at different temperatures should be moderate. In other words, after distillation accuracy of distilled model should not degrade significantly. Fig. 8 proves this point. Accuracy of the model is degraded by less than 1.38 % for all models trained at different Distillation Temperatures.

To summarize, Defensive Distillation is effective in providing resistance against adversarial examples without degrading classification correctness of model.

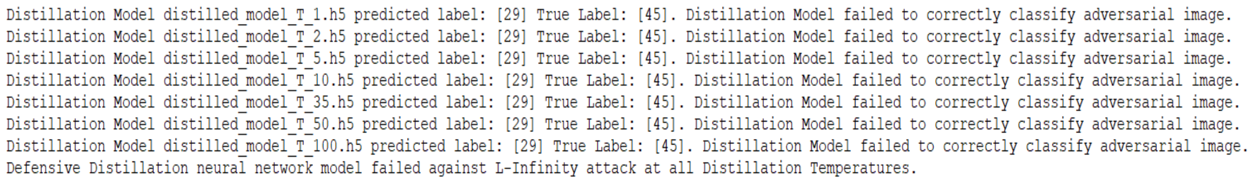
$\boldsymbol{Evaluation\ of\ L_0, L_2\ and\ L_\infty\ on\ Defensive\ Distillation\ DNN}$: Three attacks suggested by paper [5] are $L_0, L_2\ and\ L_\infty$. We have not coded these attacks ourselves, but we ran author's code using a different dataset (EMNIST dataset). Author has coded these attacks using Tensorflow 1.x. We have ported this code to Tensorflow 2.0. Instead of using the model coded by Author, we have used our own trained model to evaluate these attacks. Trained model can be found in "models" directory.

Implementation of these attacks can be found inside $main.ipynb$. To instantiate each attack, we choose a random example from EMNIST validation set. We pass this example to a particular attack API. This API returns adversarial example. We then test this adversarial example using our saved Defensive Distillation models at different Distillation Temperatures.

Attack is considered as successful only if Defensive Distillation model fails to identify the adversarial examples at all Distillation Temperatures.

$Note$: There are two variants of each attack which generates targeted and non-targeted adversarial images. We have tested both the variants. However, we have found out that targeted attacks are not always able to successfully find out an adversarial example with EMNIST dataset.

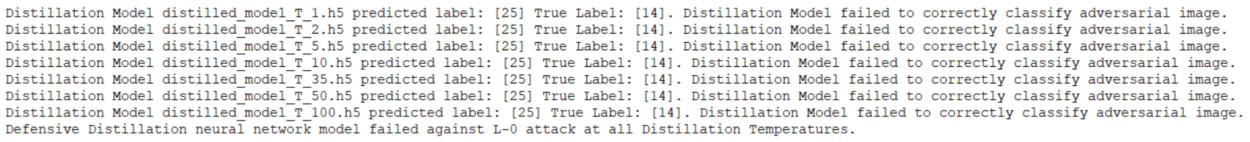
$L_\infty\ Attack$: Using this attack, we are able to generate non-targeted adversarial images which are successful in failing Defensive Distillation model at all Distillation temperatures. After successful generation of an adversarial example, our code checks if it is successful with distilled models at all values of Distillation Temperatures and outputs the results as follows:



```
Distillation Model distilled_model_T_1.h5 predicted label: [29] True Label: [45]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_2.h5 predicted label: [29] True Label: [45]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_5.h5 predicted label: [29] True Label: [45]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_10.h5 predicted label: [29] True Label: [45]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_35.h5 predicted label: [29] True Label: [45]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_50.h5 predicted label: [29] True Label: [45]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_100.h5 predicted label: [29] True Label: [45]. Distillation Model failed to correctly classify adversarial image.
Defensive Distillation neural network model failed against L-Infinity attack at all Distillation Temperatures.
```

Output clearly shows that attack is successful.

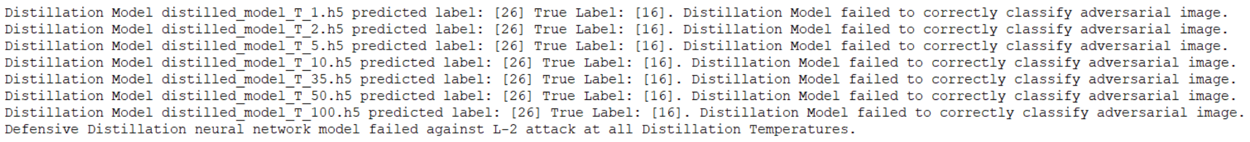
L_0 *Attack*: Using this attack, we are able to generate non-targeted adversarial images which are successful in failing Defensive Distillation model at all Distillation temperatures. After successful generation of an adversarial example, our code checks if it is successful with distilled models at all values of Distillation Temperatures and outputs the results as follows:



```
Distillation Model distilled_model_T_1.h5 predicted label: [25] True Label: [14]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_2.h5 predicted label: [25] True Label: [14]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_5.h5 predicted label: [25] True Label: [14]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_10.h5 predicted label: [25] True Label: [14]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_35.h5 predicted label: [25] True Label: [14]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_50.h5 predicted label: [25] True Label: [14]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_100.h5 predicted label: [25] True Label: [14]. Distillation Model failed to correctly classify adversarial image.
Defensive Distillation neural network model failed against L-0 attack at all Distillation Temperatures.
```

Output clearly shows that attack is successful.

L_2 *Attack*: Using this attack, we are able to generate non-targeted adversarial images which are successful in failing Defensive Distillation model at all Distillation temperatures. After successful generation of an adversarial example, our code checks if it is successful with distilled models at all values of Distillation Temperatures and outputs the results as follows:



```
Distillation Model distilled_model_T_1.h5 predicted label: [26] True Label: [16]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_2.h5 predicted label: [26] True Label: [16]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_5.h5 predicted label: [26] True Label: [16]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_10.h5 predicted label: [26] True Label: [16]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_35.h5 predicted label: [26] True Label: [16]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_50.h5 predicted label: [26] True Label: [16]. Distillation Model failed to correctly classify adversarial image.
Distillation Model distilled_model_T_100.h5 predicted label: [26] True Label: [16]. Distillation Model failed to correctly classify adversarial image.
Defensive Distillation neural network model failed against L-2 attack at all Distillation Temperatures.
```

Output clearly shows that attack is successful.

This proves an important observation pointed out in paper [5]. Paper demonstrates that, unlike other adversarial attack techniques, these three attacks are efficient and are independent of Distillation Temperature and Defensive Distillation technique is not able to eliminate adversarial images generated by these three attacks. We have proven this point by testing the adversarial image generated by each attack and by testing those adversarial images on defensive models with different Distillation Temperatures.

Summary: We have proved several claims from the papers [1], [3] and [5] using EMNIST data set as given below:

1. We have observed that as the value of ϵ increases, the adversarial sample success rate also increases which means a greater number of input images are getting classified into incorrect

output label. Here, adversarial samples are generated using FGSM technique implemented by us. This point is proven by paper [1].

2. Next, we have observed that as the distillation temperature increases, the adversarial sample success rate decreases which means that distilled network implemented by us is able to provide resilience towards adversarial examples and model's sensitivity to adversarial perturbations decreases with increase in distillation temperature. This point is proven by paper [3].
3. We have also observed that the neural network trained at different distillation temperatures has only moderate effect on the classification accuracy of the un-distilled model. This is another point proven by paper [3].
4. Finally, we have observed that $L_0, L_2\ and\ L_\infty$ attacks generate such high confidence adversarial examples that they are able to fail the defensive distillation neural network trained at different temperatures which was otherwise providing security against previous attacks but not against these three attacks. So, this clearly demonstrates the fact given in paper [5] that increasing the distillation temperature does not increase the robustness of the underlying neural network.

REFERENCES

[1] Xiaoyong Yuan, Pan He, Qile Zhu, Xiaolin Li, "Adversarial Examples: Attacks and Defenses for Deep Learning"

[2] Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy, "EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES", ICLR 2015

[3] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami, "Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks,"37th IEEE Symposium on Security & Privacy, IEEE 2016, San Jose, CA

[4] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, "Distilling the Knowledge in a Neural Network," arXiv:1503.02531v1 [stat.ML] 9 Mar 2015

[5] Nicholas Carlini, David Wagner, "Towards Evaluating the Robustness of Neural Networks," arXiv:1608.04644v2 [cs.CR] 22 Mar 2017