University of Potsdam
Research Project

ADVERSARIAL EXAMPLES FOR ROBUSTNESS OF EYE-STATE CLASSIFIER

Sefika Efeoglu
Supervised by Machine Learning Research Group
Data Science Master Program
Institute of Informatics

Abstract

The robustness of a machine learning model is quite significant for the reliability of machine learning and deep learning applications. In case the machine learning model might have enough training data, it does not have smooth model function. We set out to increase the amount of training dataset using perturbed data obtained leveraging fast gradient sign method and random noise in order to obtain a smooth model function. In addition to this, the machine learning model should be robust against noise in the real world. We also propose two approaches to deal with this noise: adversarial training algorithm and regularization-based approaches with Parseval Networks. We trained our Residual Networks with these two approaches separately and observed that increasing the amount of the training dataset with perturbed dataset had not only improved the robustness of the model on the original test dataset but also against random noise and white-box attack constructed by fast gradient sign method. Besides, the adversarial training approach has improved the model's performance against only white-box attacks. On the other hand, applying convexity and orthogonality constraints has improved the robustness of this Residual Network on the original test dataset. In this study, we utilized the small Eye-State dataset of Asaphus Vision.

Codes: `https://github.com/sefeoglu/adversarial_examples_parseval_net`

Keywords: Adversarial Examples, White-Box and Noise Attacks, Adversarial Robustness, Defence Approaches, Parseval Network.

Contents

1 Introduction

Machine learning techniques are being applied to a growing number of systems. Deep learning is a class of machine learning which uses neural network algorithms for unsupervised and supervised learning. One of the most commonly used variants is convolutional neural networks. This is often utilized in speech recognition, object detection, computer vision, natural language processing, a self-driving area with some variants of neural networks, such as RNN, CNN, GAN. One of the most important and appealing application domains is self-driving cars; for instance, deep learning techniques can assist self-driving cars in exploring the environment, helping with such things as traffic signs and surrounding objects, using images taken from cameras on the car [6].

Image classification is one of the most common computer vision problems. Some of its challenges include viewpoint variation, scale variation, deformation, intra-class variation, and background clutter [1]. Therefore, image classifiers might have misclassification problems due to noise and the natural transformation of real-world examples. In other words, when machine learning classifiers are run in real-world tasks, they are prone to fail when the training and test distributions differ. As a result, to promote their performance in real-world applications, one needs to address the lack of robustness, confidence, or uncertainty estimators, which deep learning models need to overcome.

A real-world example of this problem is safe control between vehicle and driver, which is a significant prerequisite for automated driving; whether the driver can take control or not can be evaluated using eye state detection. Such systems must be robust against changing real-world conditions, like lighting and natural transformation. These real-world conditions, which humans might not be aware of, give rise to confusion in machine learning or deep learning models, which are vulnerable to tiny perturbations in their inputs (images) [13]. These perturbations lead to misclassification problems, namely errors in the accuracy of the model.

Adversarial examples are specialized inputs created to confuse a neural network, leading to a given input's misclassification. These specialized inputs cannot be picked out by the human eye, but cause the network to fail to identify the image's contents. In addition to this, adversarial examples are intended to disturb a well-trained machine learning model. Small ad-

[1] Convolutional Neural Networks for Visual Recognition: https://cs231n.github.io/classification/

versarial perturbation must not significantly impact a trained and robust machine learning model [11]. Generating adversarial perturbation from machine learning models as negative training examples can improve the robustness of the model.

The main aim of this study is to examine the effectiveness of adding adversarial examples to the original training dataset on the performance of the model. These adversarial examples are constructed with different epsilon values from the model and added in different percentages to the original training dataset. Then, the models trained with this approach are tested on the original test dataset. In addition to this, the second aim of the project is to investigate the usefulness of three different approaches dealing with adversarial attacks, and increase the robustness and performance of a machine learning model which detects eye-states against small perturbations in an image. Three methods used in this study are adversarial training algorithms [4], regularization-based approaches (Parseval Network [4]), and adding adversarial examples to the original training dataset. In addition to this, to assess the robustness of these approaches against the attacks, we are leveraging white-box and random noise attacks.

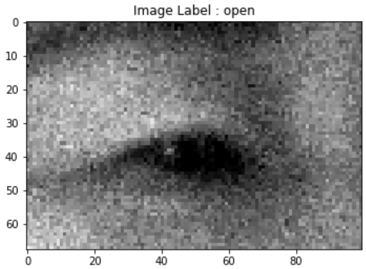


Figure 1: The illustration of an eye-state image that is labeled as “Open”.

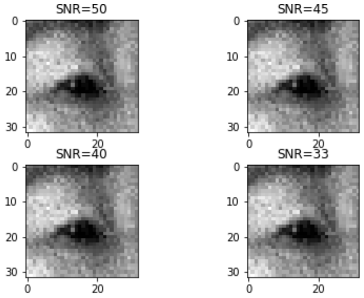


Figure 2: shows the perturbated examples of the eye-state in Figure 1, whose label is “Partially Open”.

Concerning our dataset, images of eyes consist of various eye-states labels, such as open, partially open, closed, and not visible. Natural transformations like angle of input images might lead to misclassification problems in machine learning models. The machine learning model uses sensitive measurements to recognize eye-states, so a small perturbation in an image might fool the deep learning model, as shown in Figure 1. In order to address this problem, some studies propose to add random noise to training examples.

However, these studies could not solve the problem. The most effective and straightforward approach is to create the adversarial example using the fast gradient sign method [5]. Additionally, we utilized the cleverhans library [2] in order to create the adversarial examples.

The rest of this study presents related studies on adversarial training and robustness, then sections on problem settings, background knowledge, methodology, discussion, and finally a conclusion.

2 Problem Setting

In case of small training dataset, the network model might not train enough and also might not have a smooth model function. This insufficiently trained model does not perform well on the test dataset. The accuracy of deep learning models have improved while increasing the training data (see Figure 3). The Figure 3(see the corresponding article [3] for more information) below illustrates how the performance of the deep learning model increases once the data has been increased. To increase the performance of this type of network models, the amount of the training data should be risen.

The f model in Equation 1 is not affected by this r when it is smooth enough, and it can classify the input $(x + r)$ as l label. In this case, $x + r$ is a variant of x input. To increase the smoothness of the model function, we seek whether the model can be trained by adding these negative examples like $x + r$ to the training data.

$$f(x + r) = l \; and \; f(x) = l \tag{1}$$

where x, r, l and f denote input, perturbation, label and model function, respectively. f has a continuous loss function as well.

[2] http://www.cleverhans.io

[3] https://www.kdnuggets.com/2021/06/computational-complexity-deep-learning-solution-approaches.html

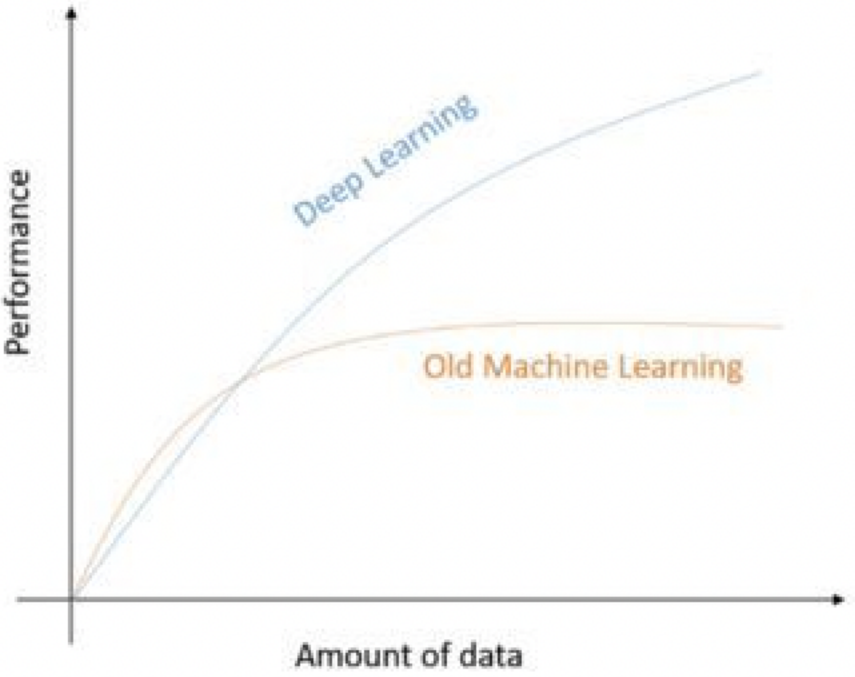


Figure 3: The performance of a deep learning model with respect to the amount of data.

3 Background Knowledge

We have utilized some metrics and methods over this work. This section gives a detailed background information about these concepts.

3.1 Adversarial Examples

Adversarial examples are inputs to machine learning models that an attacker has intentionally designed to cause the models to make a mistake; they are like optical illusions for machines. In other words, an adversarial example is an instance with small, intentional feature perturbations that cause a machine learning model to make a false prediction [4].

[4] https://christophm.github.io/interpretable-ml-book/adversarial.html

3.2 Fast Gradient Sign Method

Fast gradient sign method is a popular method to generate adversarial examples that make neural network models robust against perturbation. The output of a machine learning model is not affected by perturbation because the precision of features is limited. Nevertheless, this model is prone to misclassifying a perturbed input because of the model's softmax activation function. An adversarial example is defined as $\tilde{x} = x + \eta$.“η” is calculated by the fast gradient sign method refers to the perturbation value applied to this input (x) and the illustration of this adversarial example can be seen in Figure 4. The formula of the fast gradient sign method is below:

$$\eta = x + \epsilon \cdot sign(\triangledown_x J(\theta, x, y))$$

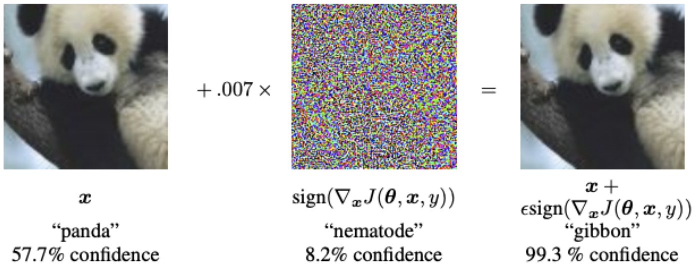

$+ .007 \times$
$=$
$\boldsymbol{x}$
“panda”
57.7% confidence
$\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“nematode”
8.2% confidence
$\boldsymbol{x} +$
$\epsilon\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“gibbon”
99.3 % confidence

Figure 4: A demonstration of fast adversarial example generation applied to GoogLeNet on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our ϵ of .007 corresponds to the magnitude of the smallest bit of an 8-bit image encoding after GoogLeNet's conversion to real numbers [5].

3.3 Signal to Noise Ratio

Signal to Noise Ratio (SNR) is a ratio of useful information to false or irrelevant data in a conversation or exchange. This SNR measurement is used in science and engineering that compares the level of the desired signal to the level of background [4]. We use SNR to interpret the results easier, like [4]. The SNR equation is

$$SNR(x, \delta_x) = 20 \log_{10} \frac{\|x\|_2}{\|\delta_x\|_2}$$

$$\delta_x = \epsilon \cdot sign(\bigtriangledown_x J(\theta, x, y))$$

for x input and perturbation (δ_x). The perturbation is computed by attack approaches like a fast gradient sign method for norm infinity.

4 Related Works

4.1 Deep Learning Models

We have exploited two different neural network architectures throughout this study.

- **Wide Residual Network (WRN)** is a solution to reduce slow training of a very deep residual network while increasing the width of the network [17]. The baseline model of WRN has 16-depth and 1-width. The weakness of very deep neural networks is high training time. The performance of this baseline model has the same accuracy as a 1000-layer thin deep network, and the number of parameters is comparable, although being several times faster to train. The widened ResNet blocks in this network allow residual networks to improves performance significantly.

- **Parseval Networks** are empirically and theoretically motivated by an analysis of the robustness of predictions made by deep neural networks when their inputs are subject to an adversarial perturbation [4]. The most important feature of Parseval Networks is to maintain weight matrices of linear convolution layers to be Parseval tight frames, which are an extension of orthogonal matrices to non-square matrices. Parseval networks are based on orthogonal and convexity constraints. Parseval networks outperform the vanilla counterpart (WRN) [4]. Meanwhile, the network has a better performance than wide residual networks against the adversarial examples having different perturbation rates.

4.2 Adversarial Attacks

A misclassification problem can be addressed by adversarial examples using various methods. These methods have been classified into two categories [8]:

- **White-Box Attacks** on a machine learning model has total knowledge about the model used for classification (e.g., type of neural network along with the number of layers) [3]. An attacker has information about the algorithm (train) used on training (e.g., gradient-descent optimization, the underlying model parameters, and architecture) and can access the training data distribution (μ) [8].

- **Black-Box Attacks** assumes no knowledge about the model and uses information about the settings or past inputs to analyze the vulnerability of the model [3]. For example, in an oracle attack, the adversary exploits a model by providing a series of carefully crafted inputs and observing outputs.

Furthermore, intriguing properties of neural networks focus on two counter-intuitive properties: the semantic meaning of individual units and the stability of neural networks [12].

Adversarial examples are robust and shared by neural nets with various layers, activations, or trained on a different subset of the training data. *Fast Gradient Sign Method (FGSM)*, a white box attack, is a popular method to generate adversarial examples that make neural network models robust against perturbations [11]. The method computes perturbation using direction (+/-) of the gradient of loss multiplied by "epsilon". Another variant of this method is *Iterative Fast Gradient Sign Method*, which extends FGSM to an iteration. Instead of applying the perturbation in a single step, it is used multiple times with a small step size [16]. Another technique to compute perturbation is *DeepFool*, which is a projection-based method [7]. The method finds the closest projection of input on hyperplane or convex Polyhedron depending on a classifier type. *Jacobian Based Saliency Map Attack (JSMA)* is a family of adversarial attack methods for fooling classification models, such as deep neural networks for image classification tasks [14]. JSMA might cause the model to misclassify resulting adversarial images as a specified erroneous target class.

Furthermore, a synthesizing technique is another approach to create adversarial examples. One of the most popular approaches in this technique is "Expected over Transportation", which changes angle and transforms the image using a chosen distribution of transformation [2]. It can be applied to two different spaces like 2D and 3D.

On the other hand, to address the misclassification problem, various regularization methods have also been investigated in the literature. One of them is Parseval Network, which is a form of deep neural networks [4]. This network approach proposed a layerwise regularization method for reducing the network's sensitivity against small perturbations by carefully controlling its global Lipschitz constant. This approach aimed to prevent the weight matrix's spectral norm by parameterizing the network with Parseval tight frames.

4.3 Adversarial Training

Adversarial Examples (AEs) have been used to improve the anti-interference ability of Artificial Intelligence models. The work proposed adversarial training to promote the robustness of the model [5]. The basic idea in this approach is to add AEs into the training data, continuously generating new AEs at each step of the training throughout the training process. The number and relative weight of AEs in each batch is controlled by the loss function independently.

A feature denoising approach for adversarial defense using Projected Gradient Descent (PGD), white-box attack, was proposed to improve the robustness of the neural networks [15]. Transformations performed by the layers in the network exhilarate the perturbation, and the hallucinated activations can overcome the activations due to the true signals, which leads the networks to make incorrect predictions. They added denoising blocks to the network to deal with this problem. However, they did not improve the accuracy when compared to the baseline models that trained with a clean dataset. Another adversarial training approach is utilizing the PGD approach and adversarial training algorithm, which is a variant of the mini-batch training algorithm [10]. Another work proposed a regularization-based approach that is faster than adversarial training and realized that by promoting linearity [9]. Therefore, the work changed the objective function using a new local linearity regularization approach.

5 Methodology

We investigate the effectiveness of adding adversarial examples to the original training dataset on the model's performance. Besides, adversarial examples have led to the fooling of the eye-state classifier, and this problem is tried to solve by using adversarial training and regularization-based approaches [4]. We examine these approaches to improve the robustness of eye-state classifiers on both the original test dataset and its adversarial examples.

5.1 Dataset and Data Preprocessing

Eye-State dataset consists of cropped images of 4 different eye-states: open, closed, partially open, and not visible as well as the distributions of the eye-states on the dataset are near to each other in Figure 5. The amount of the data in Table 1 is not enough to train the deep neural network; however, we can apply some data augmentation techniques, such as horizontal and vertical flipping, to increase a neural network's accuracy and data diversity.

Table 1: gives information about the amount of eye states on the dataset.

Eye State	**Amount**
Open	1500
Closed	1500
Partially Open	1376
NotVisible	1346

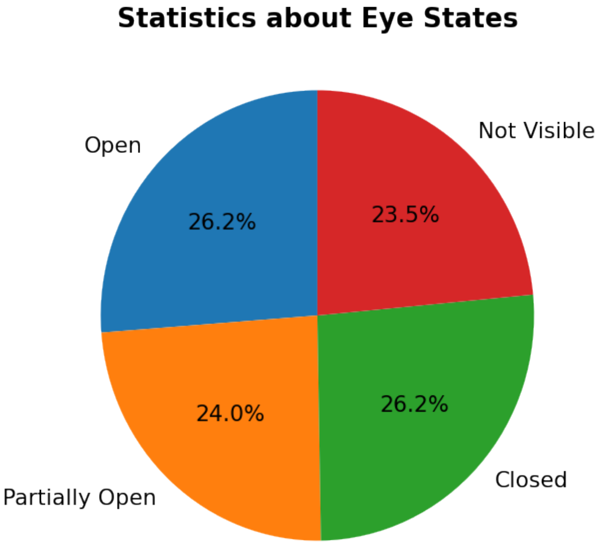


Figure 5: shows the percentages of each eye-state on the dataset.

In terms of data preprocessing, the eye-state dataset serves as cropped images, and we change only the dimensions of the input images instead of entirely data preprocessing steps. We apply the one-hot encoding technique to show labels of eye-states' images.

5.2 Neural Network Models

5.2.1 Wide Residual Network Model

We have chosen a wide residual network (WRN) structure to train the dataset, which has the highest accuracy in the previous works in the literature [17]. Additionally, we develop Parseval version of this neural network to show the effectiveness of a regularization-based approach on the original test dataset and its perturbation version. The WRN model has four different convolutional blocks. The residual block structures of this model are demonstrated in Table 1. The full model structure is in Appendix F.

Table 2: shows the structure of Wide Residual Blocks. k and N denote width and depth factors used in Wide Residual Network, respectively.

Block Name	output size	block type = B(3,3)
conv1	32 x 32	[3x3, 16]
conv2	32 x 32	$\begin{bmatrix}3x3 & 16xk \\ 3x3 & 16xk\end{bmatrix}$ x N
conv3	16 x 16	$\begin{bmatrix}3x3, 32xk \\ 3x3, 32xk\end{bmatrix}$ x N
conv4	8 x 8	$\begin{bmatrix}3x3, 64xk \\ 3x3, 64xk\end{bmatrix}$ x N
avg-pool	1 x 1	[8 x 8]

5.2.2 Parseval Network Model

Parseval Network, which is a variant of Wide Residual Network (WRN), covers convexity and orthogonality constraints defined below [4]. It utilizes orthogonal probability distribution in its kernel initializer instead of gaussian distribution like in WRN. Additionally, it has the same architecture with WRN apart from these constraints (the python codes are in Appendix C and

D). The main differences between Parseval Network and WRN are given in Table 3.

Table 3: Overview about the differences between two different networks.

Name of Property	Wide Residual Networks	Parseval Networks
Kernel Initializer	Gaussian	Orthogonal
Orthoganality Constraint	No	Yes
Convexity constraint	No	Yes

Definition 1 *Orthogonality Constraint: is an optimization algorithm on the manifold of orthogonal matrices, namely Stiefel Manifold.*

$$R_\beta\left(W_k\right) \leftarrow \frac{\beta}{2}\left|\left|W_k^T W_k - I\right|\right|_2^2,$$

where I is an identity matrix, W_k is a weight matrix (k is dimension of an input), and β is a tightness scale. Stiefel Manifold is expensive after each gradient update step.

After every main gradient update, a second update is applied to make the algorithm more efficient. Weight is computed by

$$W_k \leftarrow (1+\beta)W_k - \beta W_k W_k^T W_k$$

in this work and we assumed the β as 0.001 in this work.

Definition 2 *Convexity Constraint: In Parseval Networks, aggregation layers output a convex combination of their inputs [4]. To ensure that Lipschitz constant at the node n is such that $\Lambda_p^n \leq 1$. Euclidean projection is applied below:*

$$\alpha^* = \arg \min_{\gamma \in \Delta^{K-1}} ||\alpha - \gamma||_2^2$$

The network applies the orthogonality constraint in the convolutional blocks, while the convexity constraint is used in the aggregation layer of the network. Parseval Network model utilizes Algorithm 1 ([4]) throughout the training process, and this training algorithm utilizes these constraints, which are introduced above in Definition 1 and 2.

Algorithm 1 *Parseval Training*

```
$\Theta = \{W_k, \alpha_k\}_K^{k=1}, e \leftarrow 0$
while $\{e \leq E\}$ do
    Sample a minibatch $\{(x_i, y_i)\}_{i=1}^B$ .
    for $k \in \{1, ..., K\}$ do
        Compute the gradient
        $G_{W_k} \leftarrow \bigtriangledown_{W_k} l(\Theta, \{(x_i, y_i)\})$
        $G\alpha_k \leftarrow \bigtriangledown_{\alpha_k} l(\Theta, \{(x_i, y_i)\})$
        Update the parameters:
        $W_k \leftarrow W_k - \epsilon \cdot G_{w_k}$
        $\alpha_k \leftarrow \alpha_k - \epsilon \cdot G_{\alpha_k}$
        if hidden layer then
            Sample a set S of rows of $W_k$
            Projection:
            $W_s \leftarrow (1 + \beta)W_s - \beta W_s W_s^T W_s$.
            $\alpha_k \leftarrow argmin_{\gamma \in \Delta^{K-1}} ||\alpha_{K-\gamma}||_2^2$
        end
    end
    $e \leftarrow e + 1$
end
```

5.3 Model and Adversarial Robustness Approaches

To increase the robustness of deep learning models, we apply two different approaches. In addition, we fulfill the experiments of Parseval Networks together with an Adversarial Training approach.

1. *Adding the adversarial examples to the training dataset* fulfills the first aim of the study.

2. Adversarial training algorithm handled in collaboration with Parseval Networks in [4].

5.3.1 Training by Adding Adversarial Examples

We train Residual Network by adding adversarial examples with different perturbations and percentages to the training dataset. These adversarial examples are computed by using random noise and the fast gradient sign method. The list of epsilon values and percentages of adversarial examples used on the training of the models are [0.001, 0.003, 0.005, 0.01, 0.03] and [25, 50, 75, 100], respectively. After the training process of these models, we evaluate them on the original test dataset to investigate the improvements in their performances. In addition to this, we examine these models on the test dataset perturbed by the fast gradient sign method and random noise as well.

The learning curves of the models can be seen in Figure 12 and 15. The adversarial examples in these two figures have been computed with 0.03 epsilon value. The loss values of the models have declined throughout the training process when added the adversarial data to the training dataset. The remaining figures are in Appendix A.

5.3.2 Adversarial Training

We have exploited a mini-batch training approach to train the model adversarially [4]. For each mini-batch of the training dataset, we have changed half of the actual training data with its adversarial examples to provide stable distribution on the training dataset. These adversarial examples are computed by using the fast gradient sign method for an infinity norm. Furthermore, the epsilon factors between 0.1 and 0.03 are taken into account while computing the fast gradient sign of an image. Training of the models with these adversarial examples has resulted in a smooth model function.

Algorithm 2 Adversarial Training Algorithm

```
Inputs: model, train_dataset, epsilon_list, epochs, batch_size
Return: model
while $epochs \neq 0$ do
    $lr\_rate \leftarrow$ step_decay(epochs)
    model.optimizer.learning_rate$\leftarrow lr\_rate$
    for $(x\_train, y\_train)$ in $train\_dataset$ do
        $x\_train \leftarrow$data_augmentation(x_train, y_train, model, epsilon_list)
        model.fit(generator.flow(x_train, y_train, batch_size),batch_size)
    end
    $epochs \leftarrow epochs - 1$
end
```

The explanation of Algorithm 2:

1. *step_decay*(*epochs*) in line 4 decreases the learning rate by 0.1 rate in each ten epochs.

2. $data\_augmentation(x\_train, y\_train, model, epsilon\_list)$ function in line 7 computes the adversarial examples of the half of x_train according to epsilon_list using fast gradient sign method and combines the half of x_train with these adversarial examples.

3. $generator.flow(x\_train, y\_train, batch\_size)$ in line 8 applies 0.5 horizontal and vertical flipping to the images throughout the training.

6 Evaluation

The evaluation of a machine learning model is the most important step in which it tests the model's reliability and effectiveness. We evaluate the performances of the models trained by adding adversarial examples to the training data on the clean test data in the third subsection.

In addition to this, as aforementioned in the Methodology and Introduction sections, we have assessed the performance of three different approaches against white-box and random noise attacks. The model architectures examined in this section are Residual Network and Parseval Network, which is a variant of Residual Network. We are leveraging the signal-to-noise ratio (SNR) to easily interpret the models' performance against adversarial examples in some tables below. Low SNR indicates the highest noise in the image, which means the image has low true signal and high noise. We have trained the models using ten-fold cross-validation, which means that the accuracies reported in this section are the means obtained from ten-fold cross-validation for each model.

The rest of this section tracks hyperparameter tuning and model selection, training on Parseval and Residual Networks, the robustness of the model using the original test dataset as well as white-box and random-noise attacks.

6.1 Hyperparameter Tuning and Model Selection

In order to find the parameter settings of a model, hyperparameter tuning is applied to four different variants of the wide residual network, and each network architecture has different depth and width factors. In addition to this, we apply hyperparameter tuning for the number of the epoch, weight decay, batch size, and learning rate.

Furthermore, we have applied data augmentation to an image like 0.5 horizontal and vertical flipping during these experiments since the amount of the training dataset is quite low, as explained in the data and data preprocessing section. Ten-fold cross-validation is applied to all model types, and all performance results in this part belong to the mean of this ten fold-CV. The Stochastic Gradient Descent (SGD) optimizer is used for all models that we train in this work. We illustrate the best three models in Figure 6 and have obtained the results in Table 4 when applied grid search cross-validation for the parameters below:

- Learning Rate (LR): 0.1, 0.01.
- Weight Decay (WD): 0.01, 0.001, 0.0001.
- Batch Size (BS): 64, 128, 256.
- Epochs: 50, 100, 150.
- Width factor: 1, 2, 4.

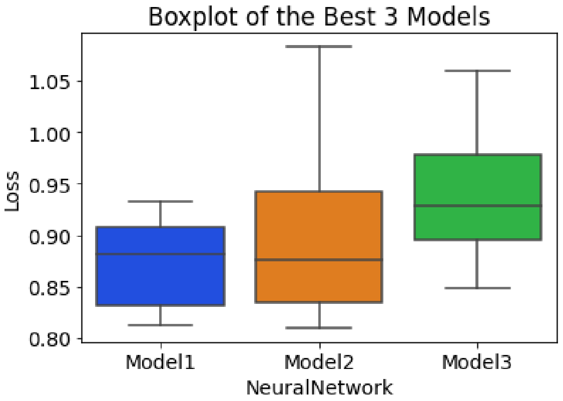


Figure 6: Hyperparameters of the best model are 0.1 learning rate, 50 epochs, 64 batch size and 0.0001 weight decay, and also k equals to 1. These results obtained using 10 Fold CV.

Table 4: Illustration of hyperparameter tuning results.

Model Number	LR	BS	# of Epoch	WD	Loss (Avg)	Accuracy (Avg)
1	0.10	50.0	64.0	0.0001	0.874353	0.707679
2	0.10	100.0	64.0	0.0001	0.899651	0.732461
3	0.01	150.0	64.0	0.0001	0.937435	0.682373

According to Table 4, Model1, which is the simple residual network (ResNet whose depth and width are 16 and 1, respectively), obtained the best results on the hyperparameter tuning experiments. The hyperparameters of this model are 0.1 learning rate, 50 epochs, 64 batch size, and 0.0001 weight decay.

On the other hand, we evaluated the effect of each hyperparameter on the performances of these best models on this hyperparameter tuning experiment. In other words, the results in the following Figures belong to the models when width factor (k) equals to 1. Figure 7 shows that the average loss has decreased when the learning rate was increased. In contrast to the learning rate, Figure 8 shows that the mean loss of the model has declined once the number of epoch was increased.

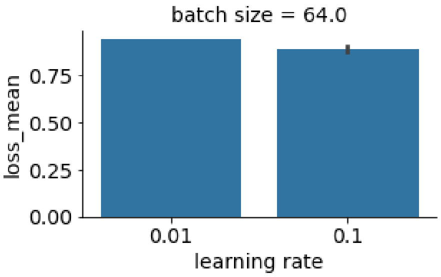


Figure 7: shows the effect of batch size and learning rate on loss means applying grid search CV.

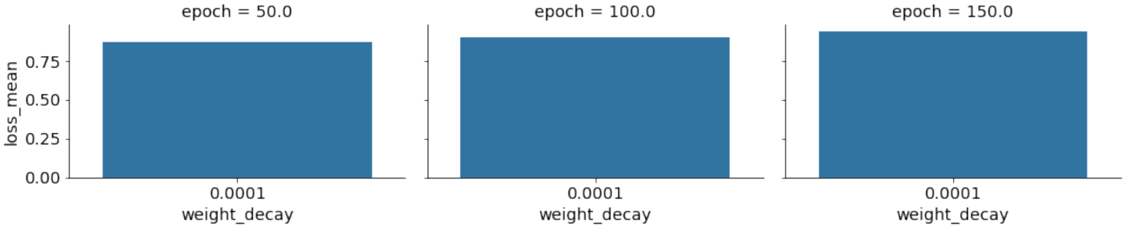


Figure 8: illustrates the effect of learning rate and epoch on mean loss throughout the hyperparameter tuning.

Finally, we have tried to increase the width factor of Wide Residual Network corresponding to the hyperparameters of the best model chosen above. Unfortunately, increasing the width factor has not improved the model's performance in terms of the mean loss (see Table 5). As a result, we did not increase the width factor.

Table 5: shows the effect of the width factor on the model's performance.

mean loss	mean acc	width factor (k)
0.874353	0.707679	1.0
1.088562	0.706981	2.0
1.253880	0.734729	4.0

6.2 Training on Parseval and Residual Networks

We have applied ten-fold cross-validation to obtain the most accurate results throughout the training of a simple WRN (ResNet) that has been chosen in the previous step. As aforementioned above, 0.5 horizontal and vertical flipping are applied to images during the training process to increase the model's robustness because our dataset is small enough. Moreover, we have used a *learning rate scheduler* that declines the learning rate with 0.1 ratio each ten epochs over the training. The reason why this learning rate scheduler has been applied is that the learning curves of the models shows a bit overfitting behavior while increasing the number of epoch and this scheduler prevented this behavior. The *learning rate scheduler* provides to work with low loss values of validation and training datasets. The loss values are computed by categorical cross-entropy. It is clear that Parseval Network has outperformed Residual Network (ResNet), which is the baseline of Parseval Network in Figures 9, 10, and 11.

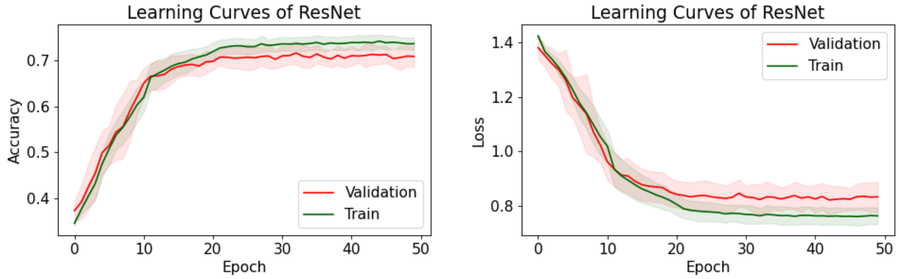


Figure 9: shows the learning curves of the ResNet model chosen in the hyperparameter tuning section.

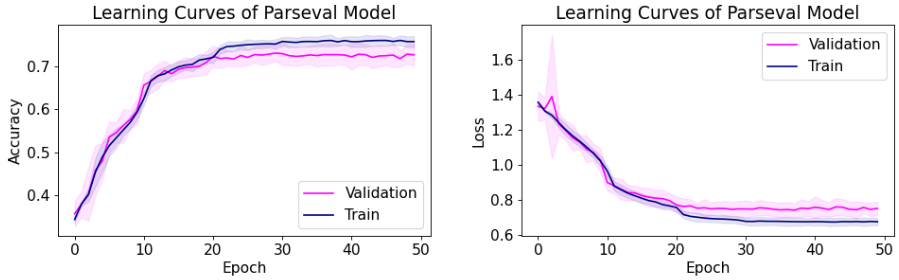


Figure 10: illustrates the learning curves of Parseval Network, which is a variant of ResNet used in the work.

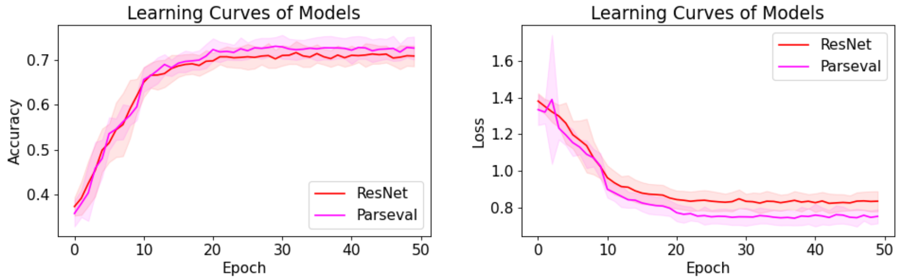


Figure 11: shows the learning curves of ResNet and Parseval Network for the comparison of their performance.

6.3 Evaluation of Models on Original Test Dataset

The accuracies of ResNet and Parseval models, which were trained with the original training dataset are 0.714 and 0.696 (the best model trained with the learning rate scheduler because of the reason mentioned in the previous section), respectively on the original test data, and Parseval outperformed ResNet in terms of model accuracy. The model accuracy has been affected somewhat by the learning rate schedular; however, the accuracy is still near to the value computed in the hyperparameter tuning. In addition to this, we trained ResNet by adding adversarial examples to the training data in different percentages. The adversarial examples have been constructed using the fast gradient sign method and random noise with different epsilon values. In the following subsections, we discuss the performances of these models.

6.3.1 Adversarial Examples using Fast Gradient Sign Method

We trained models by adding the adversarial examples with different percentages to the original training dataset. In addition to this, these adversarial examples have been created by the fast gradient sign method using different epsilon values.

The learning curves of the models can be seen in Figure 12. The adversarial examples in this Figure have been computed with 0.001 epsilon value. The loss values of the models have declined throughout the training process when added the adversarial data to the training dataset. The remaining figures are in Appendix A.

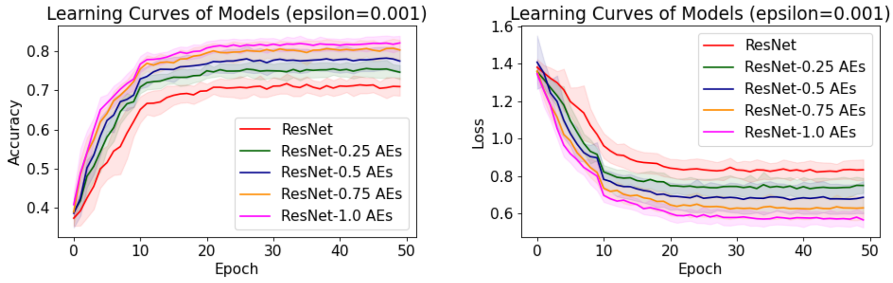


Figure 12: Learning curves of the models trained with adversarial examples constructed by Fast Gradient Sign Method. The learning curves belong to the validation data.

To examine the performances of the models, we are leveraging the micro metrics of true positive and false-positive predictions for the ten-fold cross-validation. Areas under the curves in Figure 13 and 26 provide information on how the models can distinguish eye-state labels. The performance of the models has risen steadily with the increase in the percentage of adversarial examples in the training dataset. In other words, the areas under the curves in Figure 13 and 26 have grown with the addition of more adversarial examples to the training dataset, which means that the models are able to classify eye-state labels more accurately. The accuracy of the models in Table 6 and Figure 14 has steadily increased where epsilon values equal 0.01 0.001, and 0.003, but it has not shown this behavior when epsilon values are 0.03 and 0.005. The reason why the performance has not steadily increased in these epsilon values might be the noise level in the image. The noise computed by the fast gradient sign method increased because of the epsilon value when the epsilon is 0.03. Similarly, this noise might be too small once the epsilon equals to 0.005. The remaining figures are given in Appendix B.

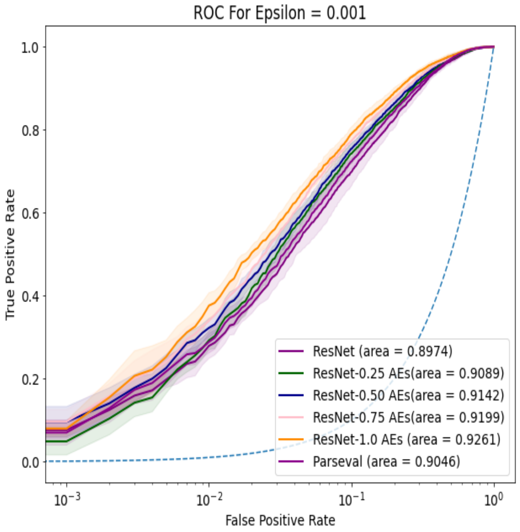


Figure 13: shows the ROC curves of ResNet, Parseval Network and ResNet models trained adding adversarial examples with different percentages (False Positive Rate is log-scale). The epsilon used in the fast gradient sign method is 0.001. The values belong to the averaged micro metrics for the ten-fold cross-validation.

The results of these experiments show that model robustness increased using the adversarial examples. The Parseval Network, a regularization-based approach, has as well increased the performance of ResNet; however, the approach of adding adversarial examples to the training dataset surpassed the Parseval Network approach with respect to accuracy.

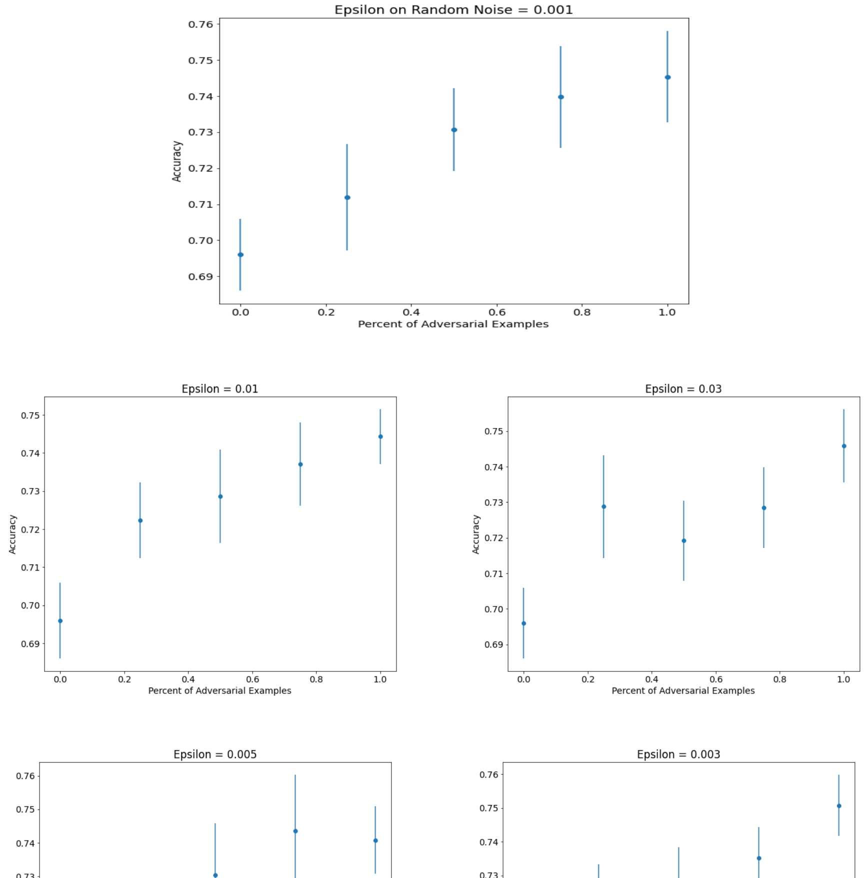


Figure 14: shows the model accuracies on the test dataset.

Table 6: The results of Residual Network models,trained adding AEs to the training dataset original test dataset.

epsilon	**% of AEs**	**mean acc**	**epsilon**	**% of AEs**	**mean acc**
0.0	0.0	0.696	-	-	-
0.001	0.25	0.711867	0.01	0.25	0.722339
0.001	0.5	0.730716	0.01	0.5	0.728621
0.001	0.75	0.739791	0.01	0.75	0.737173
0.001	1.0	0.745375	0.01	1.0	0.744328
0.003	0.25	0.720593	0.03	0.25	0.728796
0.003	0.5	0.724084	0.03	0.5	**0.719197**
0.003	0.75	0.735253	0.03	0.75	0.728447
0.003	1.0	0.750785	0.03	1.0	0.745899
0.005	0.25	0.719023	0.005	0.75	0.743630
0.005	0.5	0.730366	0.005	1.0	**0.740838**

6.3.2 Adversarial Examples using Random Noise

The adversarial examples produced by computation of the random noise using the formula in Definition 3 have been added to the training dataset throughout the training of the models. The experiments in the previous section have been repeated for this type of adversarial examples.

Definition 3 *Random Noise:*

$$\eta \sim U(-\epsilon, \epsilon)$$

$$adversarial\_example = image + \eta$$

The python code:

$$\eta = np.random.uniform(low = -\epsilon, high = \epsilon, size = image\_shape)$$

The learning curves of the models are illustrated in Figure 15. The adversarial examples in this Figure have been computed with 0.001 epsilon value. The positive impact of the method can be seen on the loss and accuracy of the models in Figure 15.

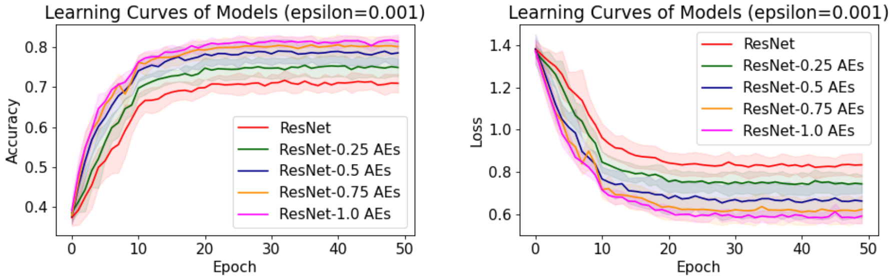


Figure 15: Learning curves of the models trained with adversarial examples constructed by random noise. The learning curves belong to the validation data.

Training the models by adding adversarial examples, created using random noise, to the training dataset has improved the robustness of the models on the original test dataset. The areas under the curves in Figure 16 and 27 have increased when the adversarial examples were added to the training dataset. However, the areas have not risen steadily with the increase in the percentage of adversarial examples in the training dataset.

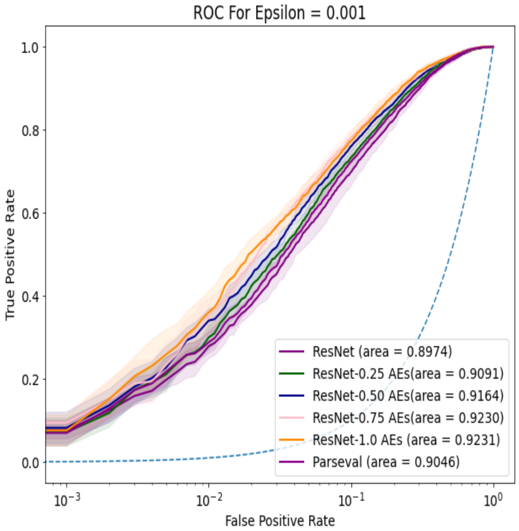


Figure 16: shows the ROC curves of ResNet, Parseval Network and ResNet models trained adding adversarial examples with different percentages (False Positive Rate is log-scale). The epsilon used in the formula of random noise is 0.001. The values based on the averaged micro metrics for the ten-fold cross-validation.

We have also checked the accuracies of these models. An increase in the accuracies of these models can be seen in Table 7 and Figure 17 when compared to the performance of ResNet trained on the original training data.

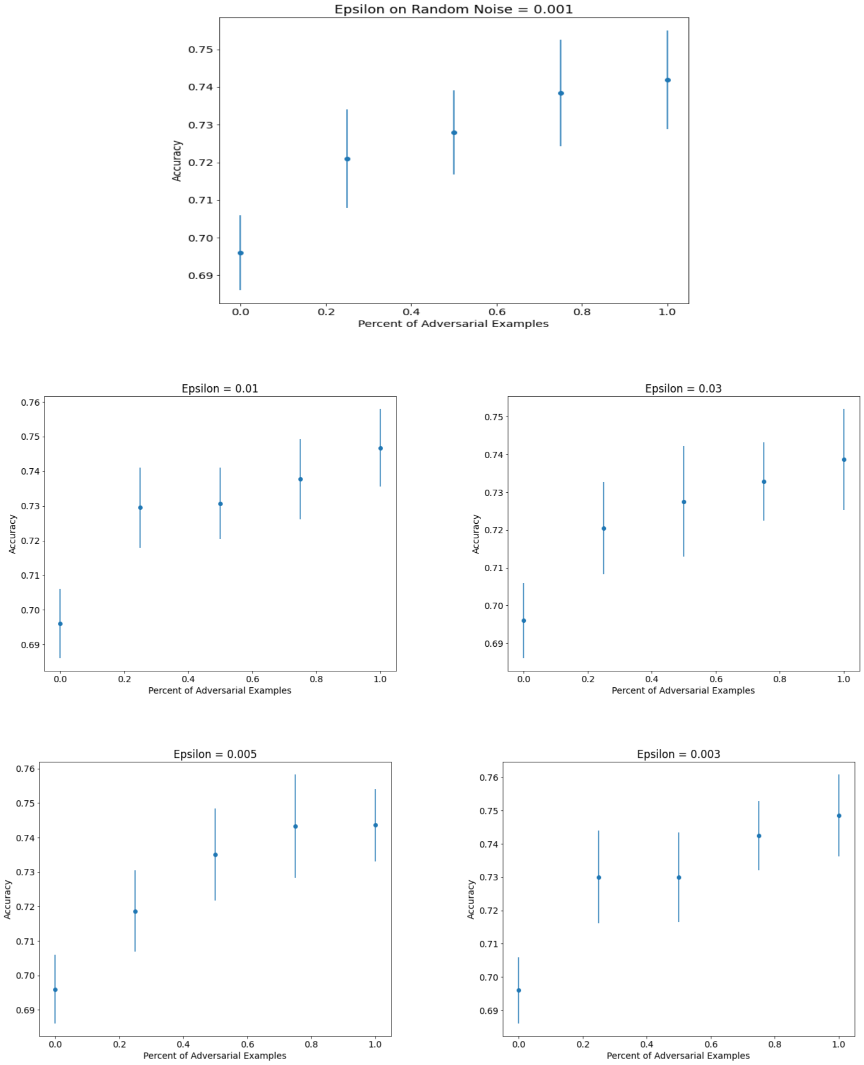


Figure 17: shows the model accuracies on the test dataset.

Table 7: shows the accuracies of the models trained using the approach on this subsection.

epsilon	**% of AEs**	**mean acc**	**epsilon**	**% of AEs**	**mean acc**
0.0	0.0	0.696	-	-	-
0.001	0.25	0.720942	0.01	0.25	0.729494
0.001	0.5	0.727923	0.01	0.5	0.730716
0.001	0.75	0.738394	0.01	0.75	0.737696
0.001	1.0	0.741885	0.01	1.0	0.746771
0.003	0.25	0.730017	0.03	0.25	0.720419
0.003	0.5	0.730017	0.03	0.5	0.727574
0.003	0.75	0.742408	0.03	0.75	0.732810
0.003	1.0	0.748517	0.03	1.0	0.738743
0.005	0.25	0.718674	0.005	0.75	0.743281
0.005	0.5	0.735079	0.005	1.0	0.743630

6.4 Whitebox Attacks

We have applied adversarial attacks to the models trained during the work and constructed adversarial examples of the test dataset using the fast gradient sign method from the cleverhans library, which is a toolbox for adversarial attacks [5].

Evaluation of Adversarial Training Approach

The ResNet and Parseval models are trained non-adversarially, whereas ResNet_ADV and Parseval_ADV are trained adversarially (Figure 18). The amount of training data is not increased with adversarial examples throughout the adversarial training approach. We have constructed the adversarial examples of the test dataset from ResNet and Parseval using the fast gradient sign method so as to attack themselves and their adversarially trained variants, respectively. Thanks to the transferability feature of adversarial examples, we have used these adversarial versions of the test dataset to attempt white-box attacks towards their adversarially trained versions, ResNet_ADV and Parseval_ADV. The accuracies of the ResNet model and its adversarially trained version (ResNet_ADV) are almost 0.25 and 0.62,

[5] http://www.cleverhans.io

respectively when evaluated using the perturbed test dataset with 0.03 epsilon, which means the adversarial training approach has increased the model robustness against its version trained without AEs in Figure 18.

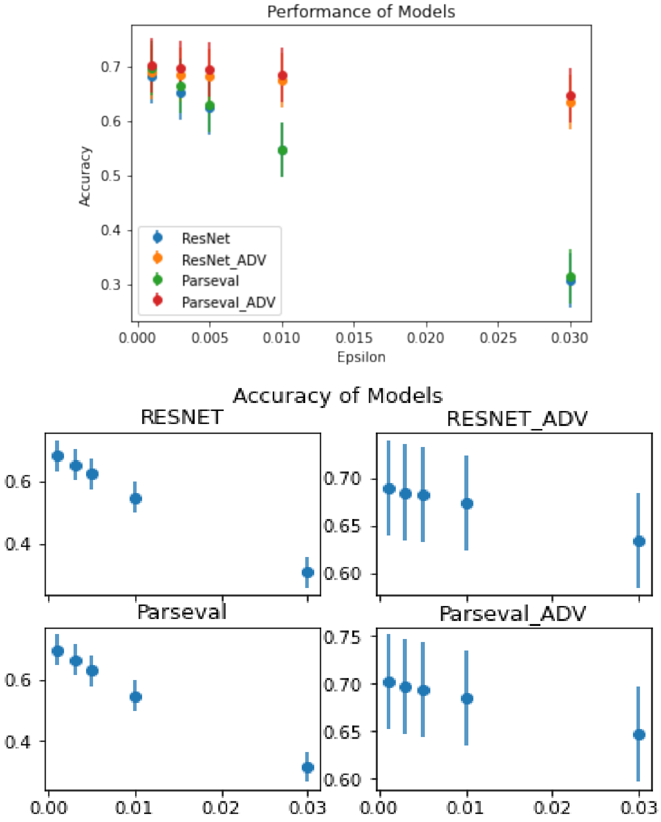


Figure 18: The accuracies of the models against the various perturbated versions of test dataset.

On the other hand, the performance of ResNet declined with the rise in the value of epsilon applied to the test dataset. On the contrary, the performance of ResNet_ADV has been mostly unaffected by this perturbation on the test dataset. Hence, it can be seen that the robustness of the model can be increased using the adversarial training approach without increasing the amount of the data with AEs in terms of white-box attacks.

Table 8: shows the performances of the models on the clean test dataset and pertubated test dataset.

Model//SNR (ϵ)	Clean	50 (0.004)	45 (0.006)	40 (0.011)	33 (0.023)
Parseval	0.714	0.665	0.629	0.562	0.4
ResNet	0.696	0.652	0.623	0.563	0.396
Parseval(Adversarial)	0.703	0.697	0.695	0.687	0.664
ResNet(Adversarial)	0.692	0.685	0.682	0.674	0.652

Evaluation of Adding Adversarial Examples to Training Dataset

The Residual Network was trained using the training dataset adding adversarial examples with different epsilons in different percentages. The models are evaluated in two ways: 1.) testing the dataset without applying the fast gradient sign method, i.e., using a clean test dataset, and 2.) testing using the perturbed version of the test dataset with various epsilon values.

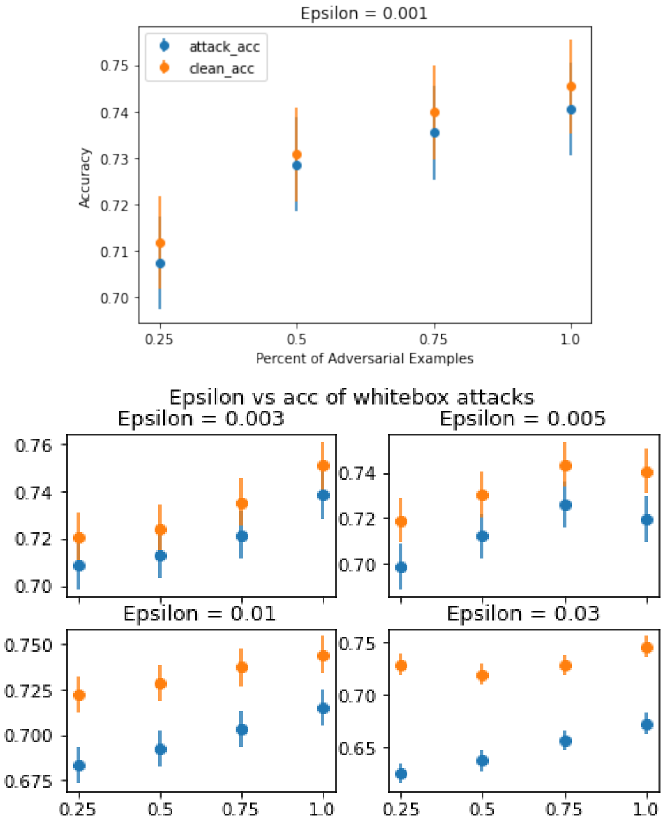


Figure 19: The illustration of the accuracy of whitebox attacks which are applied with different epsilon values.

Model accuracy against the adversarial version of the test dataset perturbated with 0.001 epsilon increased from almost 0.707 to 0.740 with the rise in the amount of the percent of adversarial examples (AEs) on the training dataset with AEs when checked Figure 19. Similarly, the accuracy of the model increased from approximately 0.711 to 0.745 using the original test dataset (clean acc in Figure 19). In Figure 19 and Table 9, the model accuracy against AEs has declined with an increase in the epsilon values. Nevertheless, the model robustness against AEs has risen while increasing the ratio of AEs on the training dataset. The model accuracy against AEs generated with 0.03 epsilon improved from 0.624 to 0.67 while increasing the rate of AEs on the training data from 0.25 to 1.0.

As a result, it is clear that adding AEs to the training dataset has not only improved the model accuracy against AEs but also improved it on the original test dataset, namely the clean test dataset.

Table 9: The results of Residual Network models,trained adding AEs to the training dataset against adversarial examples with different epsilon and original test dataset.

epsilon	**% of AEs**	**clean acc**	**attack acc**	**epsilon**	**clean acc**	**attack acc**
0.001	0.25	0.711867	0.707504	0.01	0.722339	0.683246
0.001	0.5	0.730716	0.728621	0.01	0.728621	0.692845
0.001	0.75	0.739791	0.735428	0.01	0.737173	0.703316
0.001	1.0	0.745375	0.740489	0.01	0.744328	0.714834
0.003	0.25	0.720593	0.708551	0.03	0.728796	0.624782
0.003	0.5	0.724084	0.712914	0.03	0.719197	0.636998
0.003	0.75	0.735253	0.721291	0.03	0.728447	0.656195
0.003	1.0	0.750785	0.738220	0.03	0.745899	0.672077
0.005	0.25	0.719023	0.698604			
0.005	0.5	0.730366	0.712042			
0.005	0.75	0.743630	0.726003			
0.005	1.0	0.740838	0.719721			

6.5 Random Noise Attacks

To evaluate the robustness of the models trained with two different approaches, we have attacked these models using a randomly perturbed test dataset as well. The addition of random noise to the test dataset is called a weak attack [1]. The robustness of the models against this type of weak attack provides some information about the performance of the models against real-world datasets. The random noise (η) is computed using the formula in Definition 3 above, and this noise is added to the input image. The python code of the random noise attack is given in Appendix E.

Evaluation of Adversarial Training Approach

The Parseval and ResNet models and their adversarially trained versions (Parseval_ADV and ResNet_ADV) are examined against random noise attacks. The x-axes of each figure in Figure 20 provide information about the epsilon values used in the random noise attacks, which means we applied five different random noise attacks to the models. It can be seen that the adversarial training approach does not improve the robustness of the models against random noise attacks. However, the Parseval network outperformed the Residual Network against five random noise attacks thanks to its convexity and orthogonality constraints. It is obvious that these constraints improved the robustness of ResNet (Parseval is a variant of ResNet with the addition of these constraints).

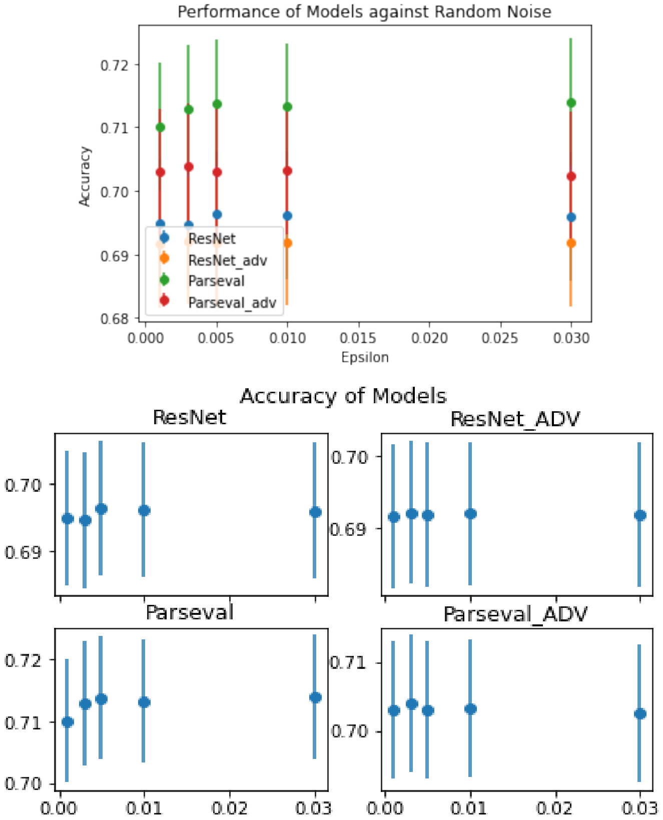


Figure 20: shows the accurancies of models against random noise attacks using different magnitudes of epsilon.

Evaluation of Adding Adversarial Examples to Training Dataset

The models were trained by adding different percentages of adversarial examples with different epsilons. In Figure 21, the epsilon values on the title of the images represent the epsilons used in the fast gradient sign method while creating adversarial examples which are added to the training dataset in different percentages.

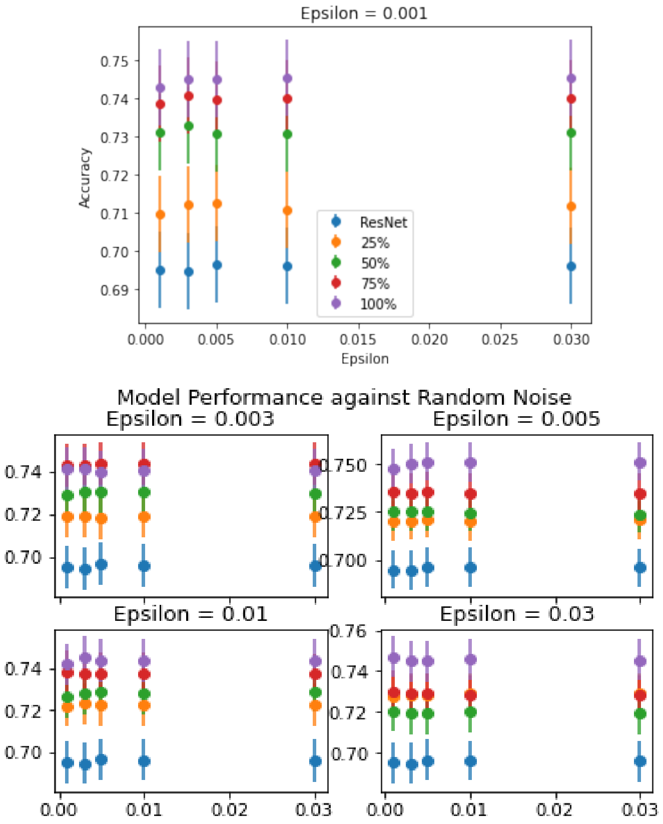


Figure 21: The performance of the models, which trained adding adversarial examples with different percent and epsilon, against the randomly perturbuted test dataset using different epsilons.

It can be seen that the ResNet model trained with the original training dataset is weak against these random noise attacks when compared with all other models. The models trained by adding adversarial examples outperformed ResNet models on various random noise attacks. However, these models do not show any pattern against these attacks, for instance, increasing accuracy with a decrease in the epsilon of the random noise. For example, the models in the second image trained with adversarial examples with 0.003 epsilon value show the different performance orders against these attacks, which means the model trained on data consisting of 75% adversarial examples shows almost the same performance as a model trained on data consisting of 100% adversarial examples. Therefore, adding adversarial examples to the training dataset has increased the model performance against random noise attacks.

As a result, we can apply this approach to increase the performance of the model (ResNet) against real-world noise in the images.

6.6 Experiments on Fully Connected Neural Networks

We have also developed a convolutional network model (CNN) with three hidden layers. We trained this model using four methods: 1.) without weight decay, 2.) with weight decay, 3.) with weight decay and an adversarial training algorithm, and 4.) with an orthogonality constraint. The model architecture covers three convolutional, input and dense layers, and also each convolutional layer is followed by one max-pooling layer and batch normalization. The model utilizes softmax activation function, and its parameters are below:

- Weight decay: 0.001
- Learning rate: 0.01
- Batch Size: 64
- Epochs: 50

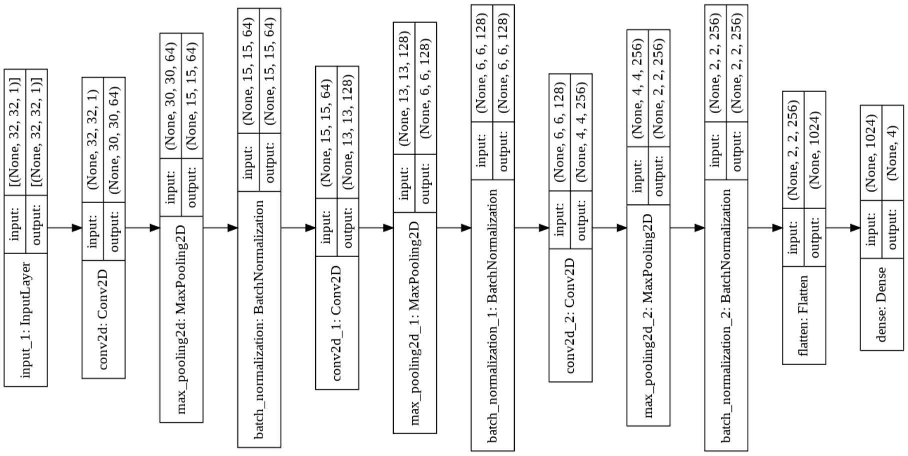


Figure 22: The illustration of the CNN model architecture.

We applied the white-box attack to these models with the same architecture, but they are trained using different approaches. The adversarial examples were created from the CNN model using a fast gradient sign method, and these adversarial examples attack the models in Figure 23. Applying weight decay to the model has increased the model accuracy against the adversarial examples. Similarly, applying the orthogonality constraint has increased the model accuracy against the adversarial examples when compared to CNN. In Figure 23, the CNN model is the baseline, and all other models are variants of this model with different configurations. CNN_WD refers to a CNN model with 0.001 weight decay. Likewise, CNN_ADV and Parseval_OC utilize an adversarial training algorithm and the orthogonality constraint, respectively, as mentioned in the Methodology section.

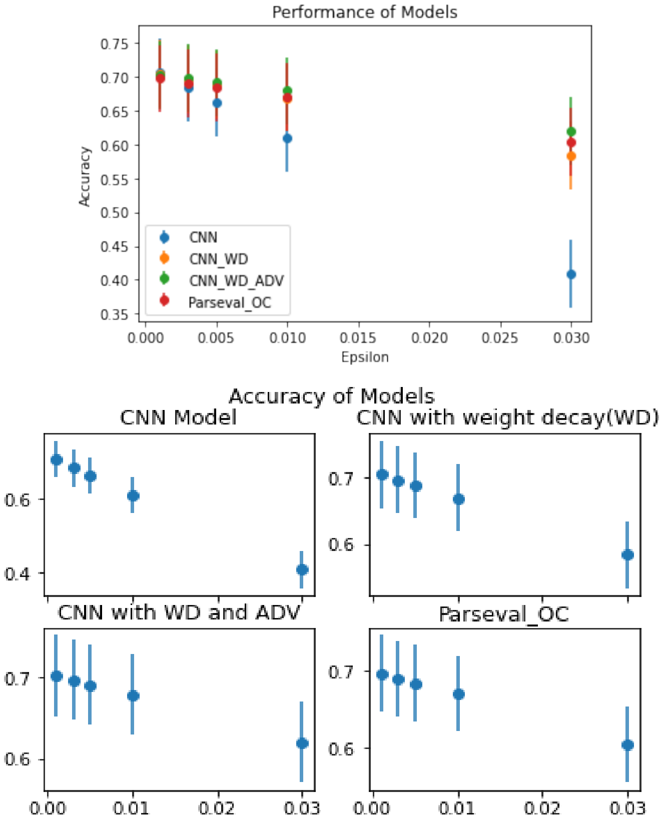


Figure 23: shows the performance results of models against white box attacks.

Table 10: shows the accuracies of fully connected models against the different SNR levels.

Model//SNR (ϵ)	Clean	50 (0.004)	45 (0.006)	40 (0.011)	33 (0.023)
CNN	0.706	0.67	0.65	0.602	0.472
CNN_WD	0.705	0.694	0.685	0.665	0.618
CNN_WD_ADV	0.703	0.695	0.689	0.677	0.642
Parseval_OC	0.6974	0.6872	0.681	0.668	0.628

7 Discussion and Future Work

The robustness of the models on the original test dataset were improved by adding the adversarial examples to the training datasets. When compared to the models' performances in terms of the adversarial example types, the models trained using the datasets consisting of adversarial examples produced by the fast gradient sign method have not performed the models trained on the dataset perturbated by random noise. Nevertheless, it is clear that increasing data diversity and amount using adversarial examples has improved the robustness of the model on the original test dataset. In addition to this, we also examined the model's performance on adversarial examples created by the fast gradient sign method from the test dataset and then repeated this experiment with random noise attacks. This approach has improved the robustness of the models against both attack scenarios.

On the other hand, we also evaluated the robustness of the models against various attacks. The adversarial training approach has shown a quite impressive performance against adversarial examples, which are created using the fast gradient sign method while not showing a similar performance against random noise attacks, as explained in Appendix C.

The question raised by this study is whether an adversarial training approach can increase the accuracy against random noise attacks. No previous study has examined random noise attacks on adversarially trained models. The study that this study was based on did not examine the models against random noise attacks [4].

8 Conclusion

The first aim of the project was to increase the robustness of the model on the original test dataset by adding adversarial examples to the training dataset. We investigated whether increasing the quantity of the training dataset using adversarial examples might improve the robustness of the model. Parseval Network approach has also improved the model accuracy on the original test dataset.

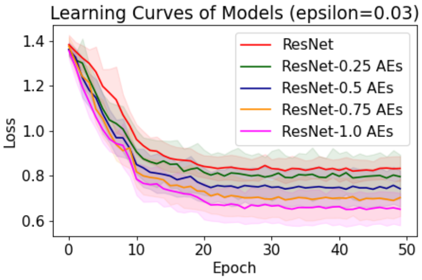
On the other hand, the machine learning model tends to make a mistake, namely misclassification, owing to small perturbations in the input. The second aim of this study was to examine the usefulness of three different approaches to overcome adversarial attacks and boost the robustness of a machine learning model, which identifies the eye-states against small perturbations in an image. We evaluated the performance of these three approaches while leveraging white-box and random noise attacks. All three approaches increased the model performance against the white-box attacks.

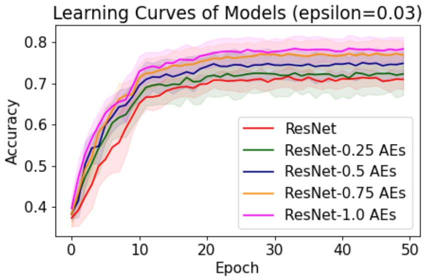
This study concludes that adding adversarial examples to the training dataset is not only a good approach to improve the performance of the model on the original test dataset but also to cope with random noise attacks known as weak attacks. Nonetheless, Parseval network has also demonstrated better performance than Residual Networks against random noise attacks. As a result of adding adversarial examples to the original dataset experiments, we can conclude that data generated by using white-box attacks like the fast gradient sign method can be used to improve model performance on the original test data. While this study did not confirm that the adversarial training algorithm is a well-performed approach for random noise attackers, it did support adversarial training as a way to increase model performance against white-box attacks.

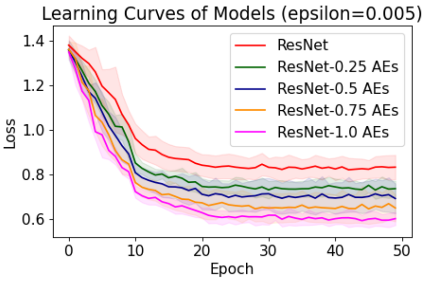
9 Appendix

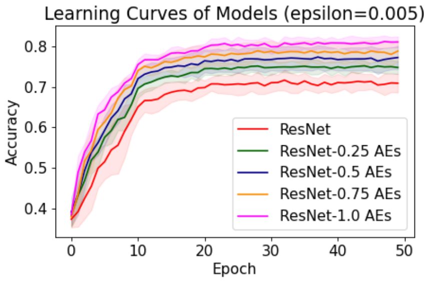
A.) Learning Curves

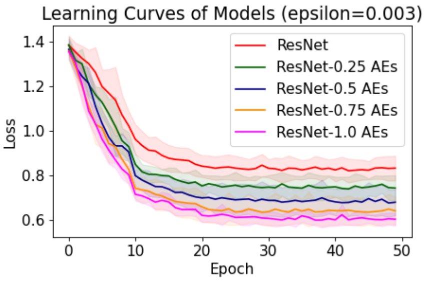
Fast Gradient Sign Method:

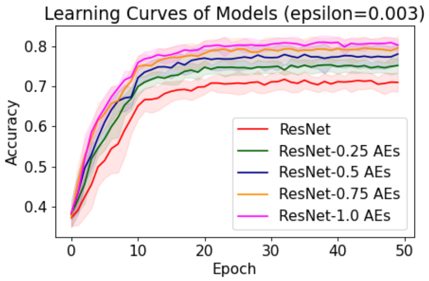












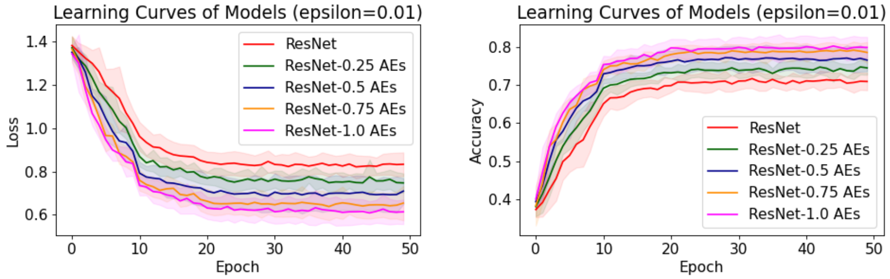


Figure 24: Learning curves of model trained with adversarial examples constructed by fast gradient sign method.

Random Noise:

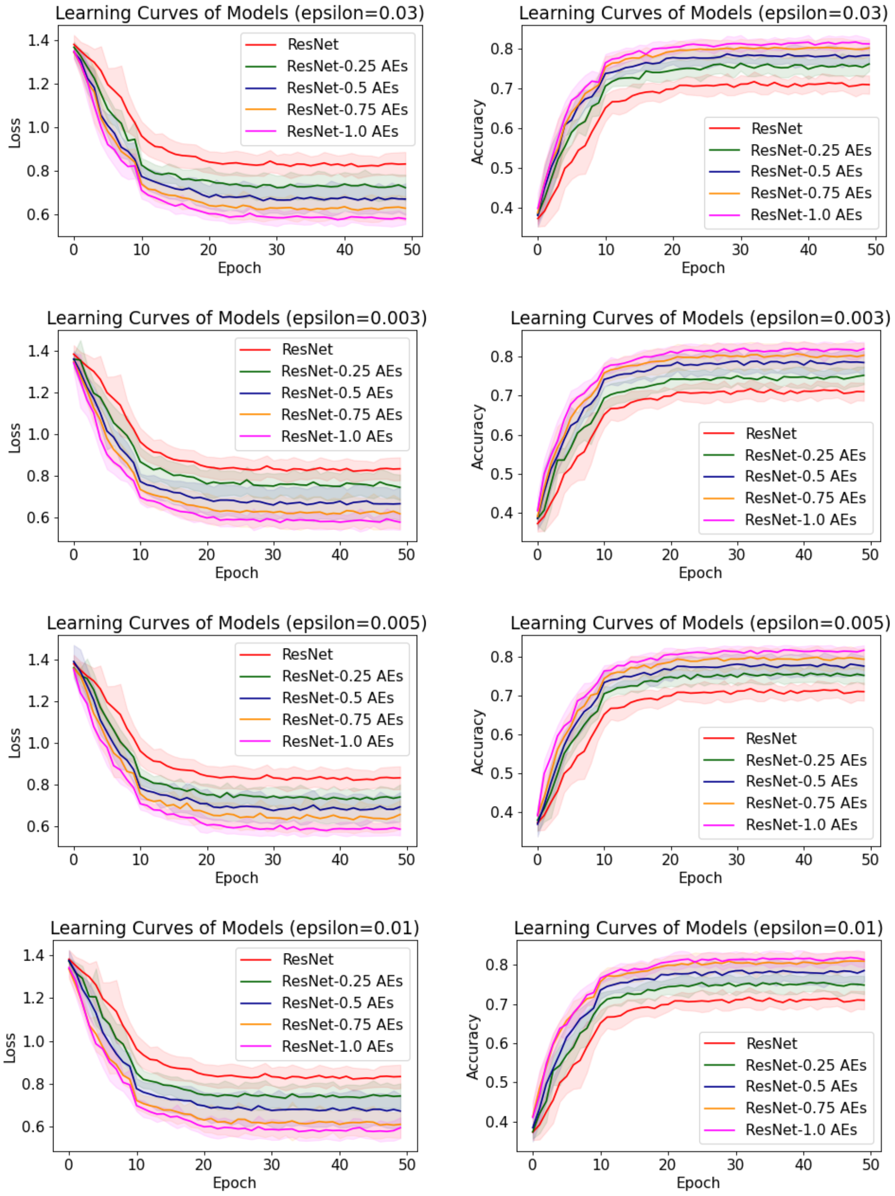


Figure 25: Learning curves of model trained with adversarial examples constructed by random noise

B.) ROC

Fast Gradient Sign Method:

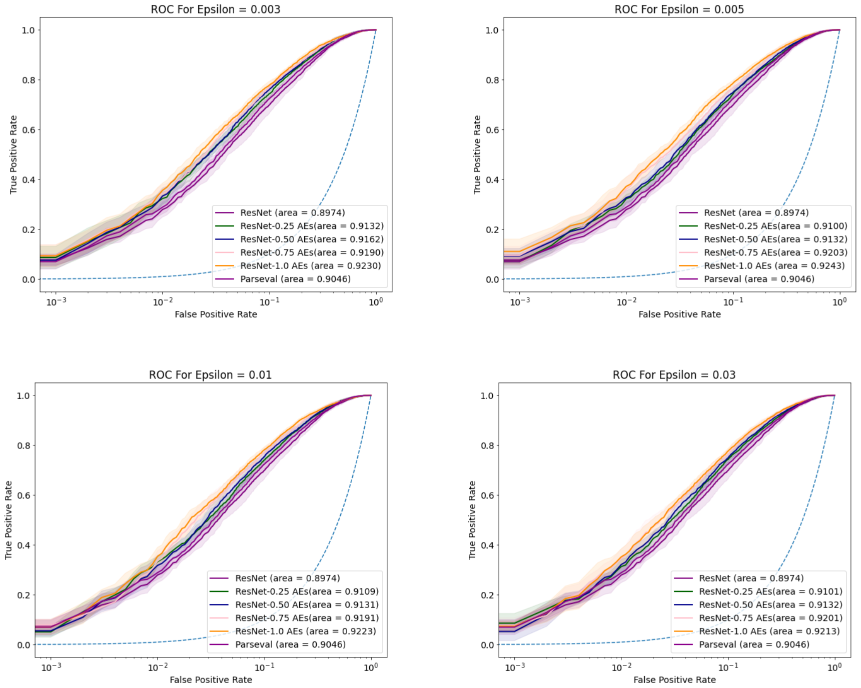


Figure 26: illustrates the ROC curves of ResNet, Parseval Network and ResNet models trained adding adversarial examples with different percentages. The epsilon values on the titles give the information about the epsilon value used in the fast gradient sign method.

Random Noise:

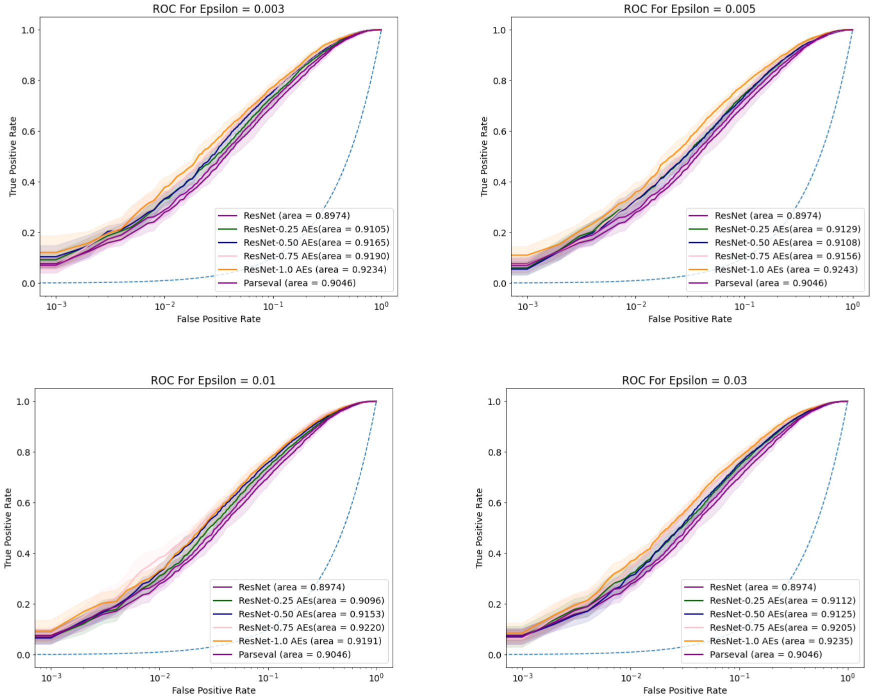


Figure 27: illustrates the ROC curves of ResNet, Parseval Network and ResNet models trained adding adversarial examples with different percentages. The epsilon values on the titles give the information about the epsilon value used in the formula of random Noise.

C.) The orthogonality constraint:

```
def orthogonality_constraint(w):
        transpose_channels = (len(w.shape) == 4)

        # Move channels_num to the front in order to make the
                                        dimensions correct for
                                        matmul
        if transpose_channels:
            w_reordered = array_ops.reshape(w, (-1, w.shape[3])
                                            )

        else:
            w_reordered = w

        last = w_reordered
        for i in range(self.num_passes):
            temp1 = math_ops.matmul(last, last, transpose_a=
                                            True)
            temp2 = (1 +
                     self.scale) * w_reordered - self.scale *
                                                     math_ops.
                                                     matmul(
                         w_reordered, temp1)
        last = temp2
        if transpose_channels:
            return array_ops.reshape(last, w.shape)
        else:
            return last
```

D.) The convexity constraint:

```
def convex_add(input_layer, layer_3, initial_convex_par=0.5,
                               trainable=False):
    """
    Do a convex combination of input_layer and layer_3. That is
                                   , return the output of

        lamda* input_layer + (1 - lamda) * layer_3


    Args:
            input_layer (tf.Tensor):Input to take convex
                                           combinatio of
            layer_3 (tf.Tensor):Input to take convex combinatio
                                            of
            initial_convex_par (float): Initial value for
                                           convex parameter.
                                           Must be in [0, 1].
            trainable (bool):Whether convex parameter should be
                                            trainable
            or not.

    Returns:
            tf.Tensor: Result of convex combination
    """
    # Will implement this as sigmoid(p)*input_layer + (1-
                                   sigmoid(p))*layer_3 to
                                   ensure
    # convex parameter to be in the unit interval without
                                   constraints during
    # optimization

    # Find value for p, also check for legal initial_convex_par
    if initial_convex_par < 0:
        raise ValueError("Convex parameter must be >=0")

    elif initial_convex_par == 0:
        # sigmoid(-16) is approximately a 32bit roundoff error,
                                        practically 0
        initial_p_value = -16

    elif initial_convex_par < 1:
        # Compute inverse of sigmoid to find initial p value
        initial_p_value = -_np.log(1/initial_convex_par - 1)

    elif initial_convex_par == 1:
        # Same argument as for 0
        initial_p_value = 16
    else:
        raise ValueError("Convex parameter must be <=1")

    p = variables.Variable(
        initial_value=initial_p_value,
        dtype=dtypes.float32,
        trainable=trainable
    )

    lam = math_ops.sigmoid(p)
    return input_layer * lam + (1 - lam)*layer_3
```

E.) Random Noise Attack

```
def noise(x, eps=0.3, order=np.inf, clip_min=None, clip_max=
                                None):
    """
    A weak attack that just picks a random point in the
                                    attacker's action
    space. When combined with an attack bundling function, this
                                     can be used to
    implement random search.
    References:
    https://arxiv.org/abs/1802.00420 recommends random search
                                    to help identify
        gradient masking
    https://openreview.net/forum?id=H1g0piA9tQ recommends using
                                     noise as part
        of an attack building recipe combining many different
                                        optimizers to
        yield a strong optimizer.
    Arguments
    ---------
    x :
        The input image.
    """

    if order != np.inf: raise NotImplementedError(ord)

    eta = np.random.uniform(low=-eps, high=eps, size=x.shape)
    adv_x = x + eta
    if clip_min is not None and clip_max is not None:
        adv_x = torch.clamp(adv_x, min=clip_min, max=clip_max)

    return adv_x
```

F.) Residual Network Structure

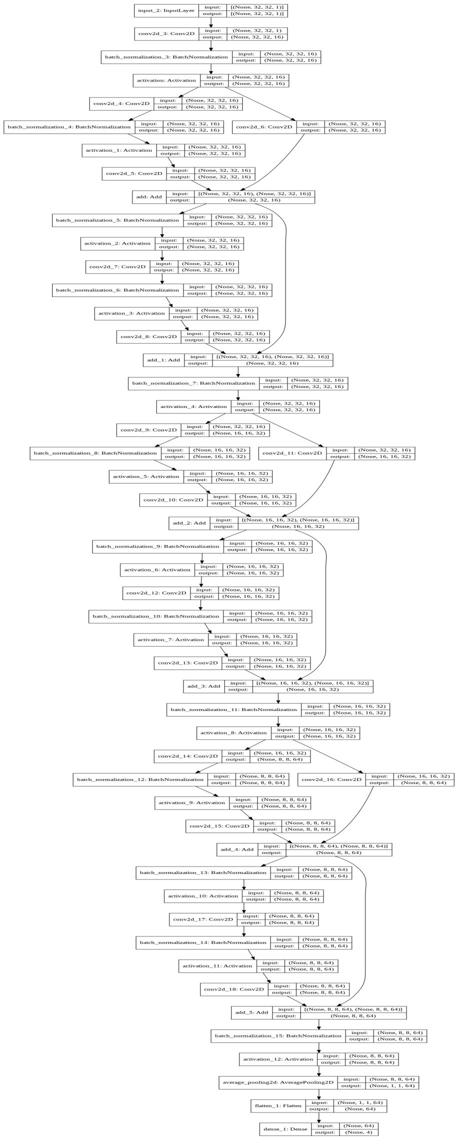

input_2: InputLayer input: [(None, 32, 32, 1)] output: [(None, 32, 32, 1)]
conv2d_3: Conv2D input: (None, 32, 32, 1) output: (None, 32, 32, 16)
batch_normalization_3: BatchNormalization input: (None, 32, 32, 16) output: (None, 32, 32, 16)
activation: Activation input: (None, 32, 32, 16) output: (None, 32, 32, 16)
conv2d_4: Conv2D input: (None, 32, 32, 16) output: (None, 32, 32, 16)
batch_normalization_4: BatchNormalization input: (None, 32, 32, 16) output: (None, 32, 32, 16)
conv2d_6: Conv2D input: (None, 32, 32, 16) output: (None, 32, 32, 16)
activation_1: Activation input: (None, 32, 32, 16) output: (None, 32, 32, 16)
conv2d_5: Conv2D input: (None, 32, 32, 16) output: (None, 32, 32, 16)
add: Add input: [(None, 32, 32, 16), (None, 32, 32, 16)] output: (None, 32, 32, 16)
batch_normalization_5: BatchNormalization input: (None, 32, 32, 16) output: (None, 32, 32, 16)
activation_2: Activation input: (None, 32, 32, 16) output: (None, 32, 32, 16)
conv2d_7: Conv2D input: (None, 32, 32, 16) output: (None, 32, 32, 16)
batch_normalization_6: BatchNormalization input: (None, 32, 32, 16) output: (None, 32, 32, 16)
activation_3: Activation input: (None, 32, 32, 16) output: (None, 32, 32, 16)
conv2d_8: Conv2D input: (None, 32, 32, 16) output: (None, 32, 32, 16)
add_1: Add input: [(None, 32, 32, 16), (None, 32, 32, 16)] output: (None, 32, 32, 16)
batch_normalization_7: BatchNormalization input: (None, 32, 32, 16) output: (None, 32, 32, 16)
activation_4: Activation input: (None, 32, 32, 16) output: (None, 32, 32, 16)
conv2d_9: Conv2D input: (None, 32, 32, 16) output: (None, 16, 16, 32)
batch_normalization_8: BatchNormalization input: (None, 16, 16, 32) output: (None, 16, 16, 32)
conv2d_11: Conv2D input: (None, 32, 32, 16) output: (None, 16, 16, 32)
activation_5: Activation input: (None, 16, 16, 32) output: (None, 16, 16, 32)
conv2d_10: Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 32)
add_2: Add input: [(None, 16, 16, 32), (None, 16, 16, 32)] output: (None, 16, 16, 32)
batch_normalization_9: BatchNormalization input: (None, 16, 16, 32) output: (None, 16, 16, 32)
activation_6: Activation input: (None, 16, 16, 32) output: (None, 16, 16, 32)
conv2d_12: Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 32)
batch_normalization_10: BatchNormalization input: (None, 16, 16, 32) output: (None, 16, 16, 32)
activation_7: Activation input: (None, 16, 16, 32) output: (None, 16, 16, 32)
conv2d_13: Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 32)
add_3: Add input: [(None, 16, 16, 32), (None, 16, 16, 32)] output: (None, 16, 16, 32)
batch_normalization_11: BatchNormalization input: (None, 16, 16, 32) output: (None, 16, 16, 32)
activation_8: Activation input: (None, 16, 16, 32) output: (None, 16, 16, 32)
conv2d_14: Conv2D input: (None, 16, 16, 32) output: (None, 8, 8, 64)
batch_normalization_12: BatchNormalization input: (None, 8, 8, 64) output: (None, 8, 8, 64)
conv2d_16: Conv2D input: (None, 16, 16, 32) output: (None, 8, 8, 64)
activation_9: Activation input: (None, 8, 8, 64) output: (None, 8, 8, 64)
conv2d_15: Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 64)
add_4: Add input: [(None, 8, 8, 64), (None, 8, 8, 64)] output: (None, 8, 8, 64)
batch_normalization_13: BatchNormalization input: (None, 8, 8, 64) output: (None, 8, 8, 64)
activation_10: Activation input: (None, 8, 8, 64) output: (None, 8, 8, 64)
conv2d_17: Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 64)
batch_normalization_14: BatchNormalization input: (None, 8, 8, 64) output: (None, 8, 8, 64)
activation_11: Activation input: (None, 8, 8, 64) output: (None, 8, 8, 64)
conv2d_18: Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 64)
add_5: Add input: [(None, 8, 8, 64), (None, 8, 8, 64)] output: (None, 8, 8, 64)
batch_normalization_15: BatchNormalization input: (None, 8, 8, 64) output: (None, 8, 8, 64)
activation_12: Activation input: (None, 8, 8, 64) output: (None, 8, 8, 64)
average_pooling2d: AveragePooling2D input: (None, 8, 8, 64) output: (None, 1, 1, 64)
flatten_1: Flatten input: (None, 1, 1, 64) output: (None, 64)
dense_1: Dense input: (None, 64) output: (None, 4)

Figure 28: Residual Network Architecture whose width and depth are 16 and 2, respectively.

References

[1] Athalye, A., Carlini, N., Wagner, D.: Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples (2018)

[2] Athalye, A., Engstrom, L., Ilyas, A., Kwok, K.: Synthesizing robust adversarial examples (2018)

[3] Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., Mukhopadhyay, D.: Adversarial attacks and defences: A survey (2018)

[4] Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., Usunier, N.: Parseval networks: Improving robustness to adversarial examples (2017)

[5] Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2015)

[6] Liu, C.: Adversarial deep learning for autonomous driving, `https://deepdrive.berkeley.edu/project/adversarial-deep-learning-autonomous-driving`

[7] Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks (2016)

[8] Nazemi, A., Fieguth, P.: Potential adversarial samples for white-box attacks (2019)

[9] Qin, C., Martens, J., Gowal, S., Krishnan, D., Dvijotham, K., Fawzi, A., De, S., Stanforth, R., Kohli, P.: Adversarial robustness through local linearization (2019)

[10] Shafahi, A., Najibi, M., Ghiasi, A., Xu, Z., Dickerson, J., Studer, C., Davis, L.S., Taylor, G., Goldstein, T.: Adversarial training for free! (2019)

[11] Sun, S., Yeh, C.F., Ostendorf, M., Hwang, M.Y., Xie, L.: Training augmentation with adversarial examples for robust speech recognition (2018)

[12] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2014)

[13] Tanay, T.: Adversarial examples, explained, `https://www.kdnuggets.com/2018/10/adversarial-examples-explained.html`

[14] Wiyatno, R., Xu, A.: Maximal jacobian-based saliency map attack (2018)

[15] Xie, C., Wu, Y., van der Maaten, L., Yuille, A., He, K.: Feature denoising for improving adversarial robustness (2019)

[16] Xie, C., Zhang, Z., Zhou, Y., Bai, S., Wang, J., Ren, Z., Yuille, A.: Improving transferability of adversarial examples with input diversity (2019)

[17] Zagoruyko, S., Komodakis, N.: Wide residual networks (2017)