Division of Electronics and Information Engineering

FINAL ANALYSIS REPORT

ADVANCED IMAGE PROCESSING (COMPUTER VISION)

December 8th, 2020

Reported by: Vuong Minh Phu (ID: 202050798)

Instructor: Prof. Hyojong Lee

Table of contents

Introduction

With the rise of machine learning and deep learning techniques, problems in many fields can be solved easily, such as autonomous driving cars, voice command recognition, competitive gaming (Alpha GO). One of the most important applications of these techniques is image processing. By using neural networks, especially convolutional neural networks, extracting images features becomes easier and more accurate. Interestingly, in 2015, neural networks surpassed human accuracy in image classification tasks on ImageNet dataset, marking a crucial milestone in computer visual system development.

However, it turns out these networks are easy to break with perturbations. Specifically, when we add a small amount of noises to the images, which cannot be recognised by human perception, the accuracy of the neural networks drops down significantly. We call these images adversarial examples [1]. Using these examples, we can cause inconsistencies to very powerful models. Therefore, it is necessary to truly understand neural networks, or at least able to develop methods to mitigate these kinds of problems. One possible solution could be to train the models on adversarial examples in order to increase the robustness of them against adversarial attacks.

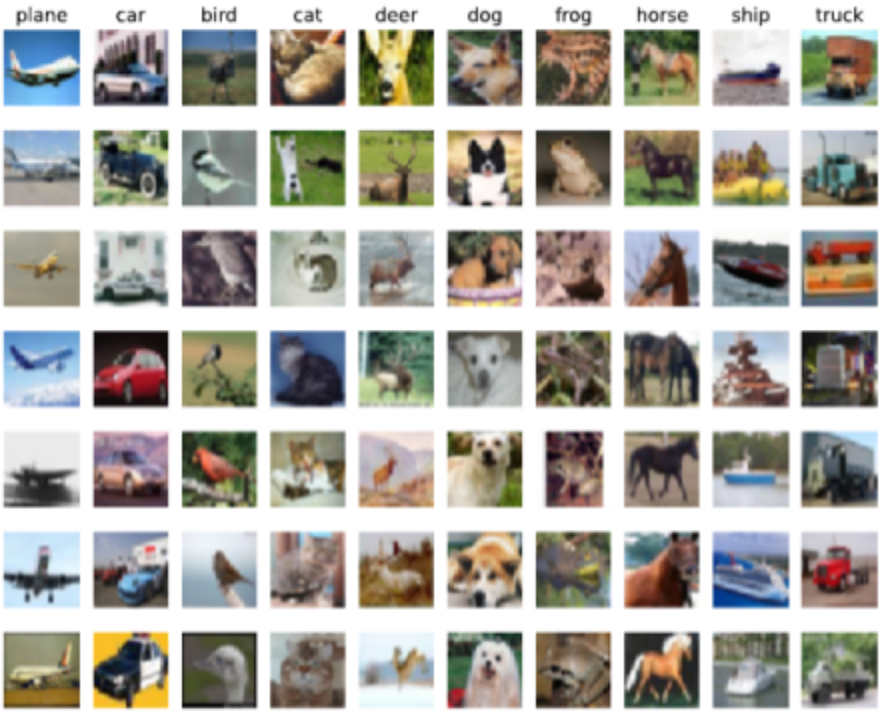
In this report, I will first summarise the development of neural networks and compare their accuracy with classical image processing technique. Then I show the inconsistency of these networks by performing adversarial attacks on them. Finally, I introduce some methods that could be used to combat this situation.

Problem statement

To compare neural networks with handcraft mathematical methods, I will use a very powerful model called Convolutional Neural Networks, particularly ResNet [2], and a Multiclass Support Vector Machine [3], to solve image classification tasks. The dataset I use is CIFAR-10 [4] which consists of 10 classes namely [plane, car, bird, cat, deer, dog, frog, horse, ship, truck]. Given an image in the dataset, the models need to correctly classify the input image into the true class.

These two models then will be evaluated and attacked by using Projected Gradient Descent method [6], a variation of Fast Gradient Sign Method [7], and re-evaluated again after training them on adversarial examples. I will also introduce how I can process the data as well as the training step for each of the models.

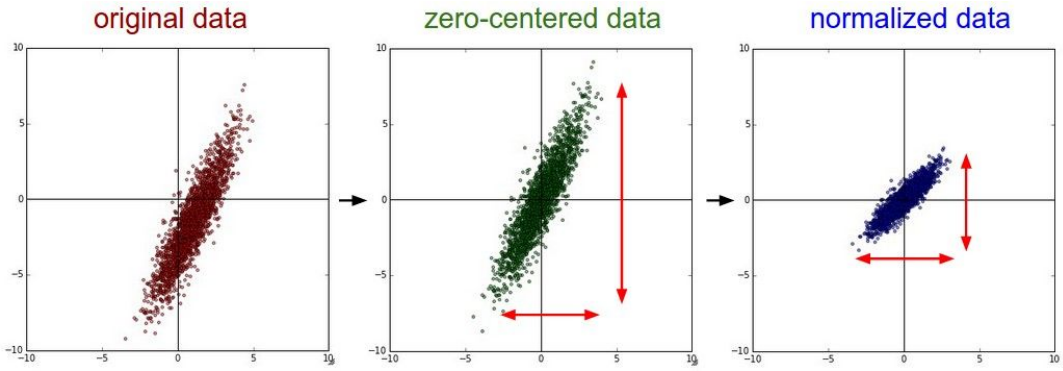
Data processing



CIFAR-10 dataset

CIFAR-10 dataset consists of 50,000 training and 10000 testing data. Each of them are a 32x32 RGB image, and a one-hot vector of labels. Firstly, I divide the training data into training set and validation set with the proportion of 90-10. It is important to do this because we want to evaluate our model on unseen data before testing it. In other words, we are not going to touch the test set until we finish training our model.

Secondly, we flatten out our image into a single vector and normalize our data to the range of (0, 1) instead of (0, 255). We also need to subtract the image mean across 50,000 training data. The reason we do both of these techniques is because in the process of training our networks, the data must go through different layers, or multiply them with the weights, plus the bias and use some non-linearity functions on them. If we do not set them in the right range, it is hard to control their value. In other words, training them with neural networks involves gradient descent and backpropagation algorithms, and if our data are not in control, calculating the parameters for the former methods will be very difficult.



Data processing pipeline. **Left** original 2D data. **Middle:** data is zero-centering by subtracting the mean in each dimension. **Right:** normalized by scaling the images by its standard deviation.

After preprocessing the data, we are good to use the models to classify it. Note that the training data will be divided into 90% training and 10% for validation.

Gradient descent

Loss function

Because I will use gradient descent in training for both ResNet and SVM, it is important to introduce it briefly. To begin, we need two key components in context of image classification which are score function and loss function. To be more specific, a score function is a linear function that maps the raw image pixels into the correct class scores, while a loss function measures how well the score function performs on a given dataset. For example, a Multiclass Support Vector Machine can be seen as a loss function, while a neural network is a combination of a score function and a loss function.

Interpretation of SVM loss

A score function has a form: $f(x_i, W) = y'$ where x_i is an input image, W is a set of weights and y' is a predicted output from our model. A loss function can be represented as: $L(x, W, y)$. There are many form of the function L, but we will only discuss some of them in the scope of this report.

Multiclass SVM or Hinge loss:

$$L_{SVM}(x,\ W, y) = \frac{1}{N}\sum_{i}\sum_{j\neq yi}\left[max(0, f(x_i, W)_j - f(x_i, W)_{yi} + \Delta)\right] + \lambda\sum_{k}\sum_{l} W_{k,l}^2$$

where the first term indicates the data loss and the second term represents the regularization loss. The intuition behind SVM loss is that it want the correct class for each image to have a score higher than the incorrect classes by some fixed margin Δ

Softmax function or log loss:

$$L_i = -log\left(\frac{e^{f_{yi}}}{\sum_j e^{f_j}}\right).$$

The general idea of softmax is it will take a vector of scores and squashes it into a vector values between zero and one. These values represent the probabilities of a predicted class over the rest of the class in the dataset.

Gradient descent

Now we have the loss functions, to train the models, our goal is to minimize these functions. In order to optimize our functions, we can find direction toward the minimum. The process of finding the direction is called compute descent. It turns out we can find the best direction by computing the partial derivatives of a loss function with respect to the weight parameter.

Gradient descent is a repeated process of calculating gradients and performing parameters update.

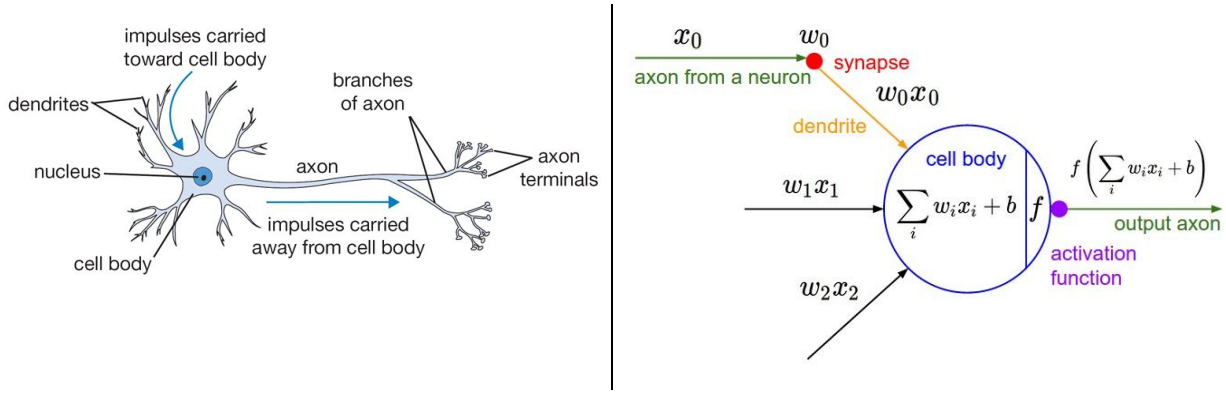
For example, we can calculate the gradient of the SVM loss as follow:

$$\frac{dL_i}{dW_{yi}} = -\left(\sum_{j\neq yi} 1(w_j^T x_i - w_{yi}^T x_i + \Delta > 0)\right) x_i$$

$$\frac{dL_i}{dW_j} = 1(w_j^T x_i - w_{yi}^T x_i + \Delta > 0) x_i$$

Note that 1 is the indicator function, resulting in one if the condition inside is true or zero otherwise.

Neural Networks



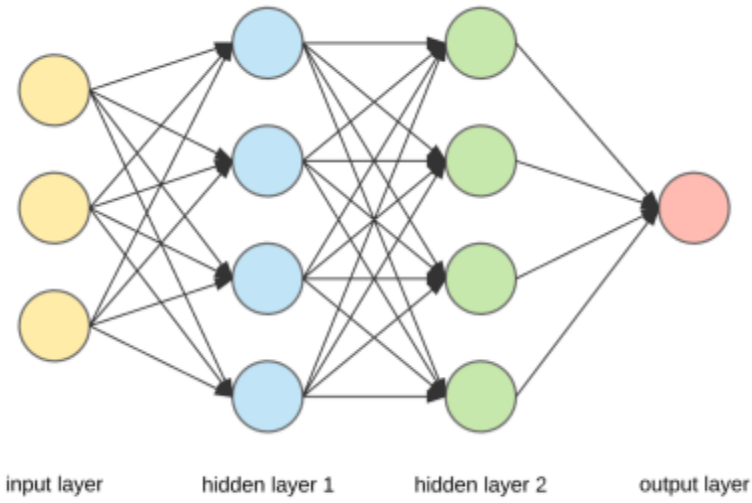
Neural Network: Biological neuron (left) and mathematical model (right)

Neural network is the model that mimics the human neuron analogy. Basically, each neuron will take the input signals from the other neurons and output to the neurons sequentially. Using this topology, we can interpret neural networks as multiple layers of neurons which take images as inputs and produce output as the probabilities map or scores in general. For examples, a three layers of neural networks can be represented as follow:

$$y = W_3 f_2(W_2, f_1(W_1, x)),$$

where f_i , $i = 1, 2, 3$ is called activations which provide the non-linearity. There are a wide range of activations, but the the most popular one is ReLU [5] or rectified linear unit:

$$ReLU(x) = max(0, x).$$



Three layers neural network

Using gradient descent to minimize the loss function we can train our neural networks to fit the data. Surprisingly, practice shows that these models are very good at almost every task ranging from image classification, voice recognition, visual tracking, human pose estimation or even language modelling.

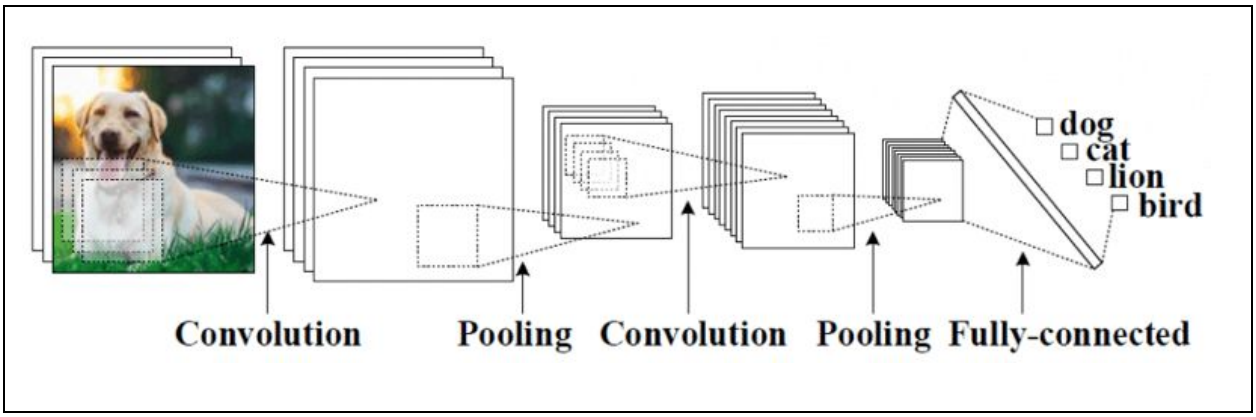
Convolutional neural networks (CNNs)

ConvNet

There are many kinds of neural networks, but for the sake of image classification, it is shown that CNNs perform the best on this task. One reason this kind of network is preferable over plain neural nets is that it scales well to full image. For example,when we input an image with size 32x32x3 in the case of CIFAR-10, the first fully-connected layer would have 32*32*3 = 3072 weights. This amount seems manageable, but as we go deeper, the number will strike up leading to computational burden.

In contrast, ConvNet arranges the layers in 3 dimensions: width, height, depth, instead of squashing the data into a single vector. A neuron in a layer will connect to a small portion of neurons in the previous layer rather than connect in a fully-connected mechanism. Moreover, the last layers of CNNs will take advantage of fully-connected layers to produce accurate output.

The basic blocks building CNNs are **convolutional layer, pooling layer** and **fully-connected layer.** These layers will be stacked together to form a convolution neural network architecture. See the illustration below.



Convolutional neural networks architecture: Constructed by sequential connecting convolution, pooling and fully connected layers together.

Convolution layer

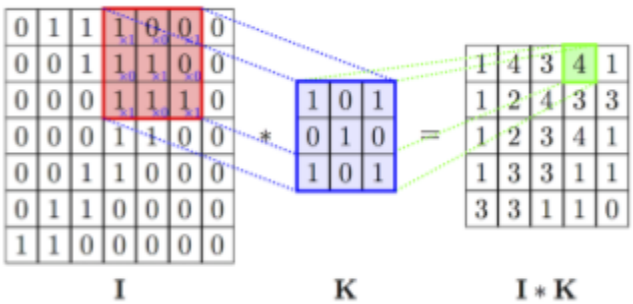
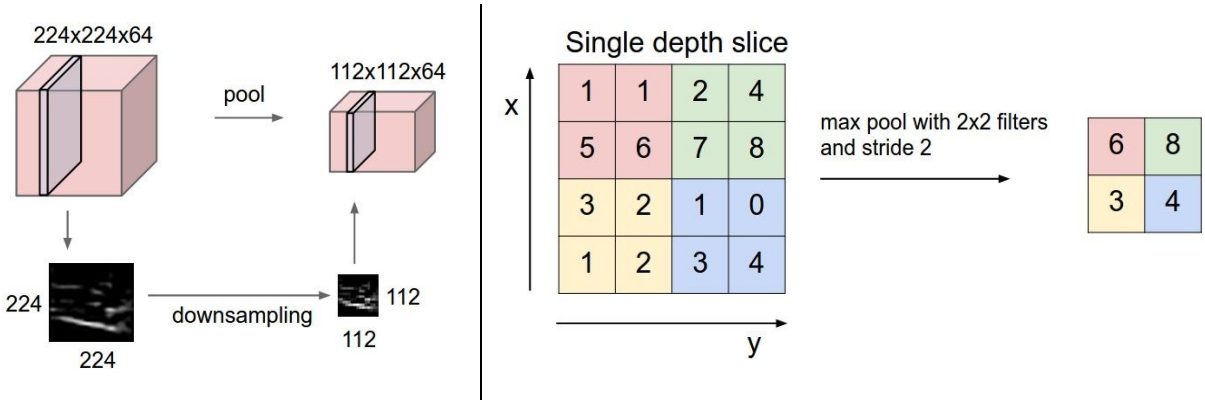


Illustration of convolution layer

The principle of Conv layer is the application of a filter over an input multiple times which results in a stack of features at the output. These features will first go through an activation function (usually ReLU) before feeding into the pooling layer. As we discussed, instead of fully connecting to every neuron in the previous layer, each layer in ConvNet connects to some local regions of them which we call the receptive field. Each computation consists of a dot product of the filter weights and a small amount of data in the previous layer. This calculation is just like the classic filter in image processing, but the weights of filters which can be learned by training.

Pooling layer

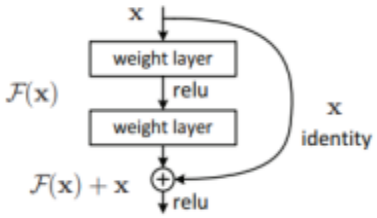
It is common to use a pooling layer between every convolution layer in ConvNet. It is beneficial to insert them due to the reduction in computational resources, hence control overfitting. Generally, a pooling layer will help reduce the size of the input feature by two, when conserving the depth(or channel).



Pooling layer illustration: most common downsampling operation is max pooling where the max is taken over a local filter size

ResNet

There are many kinds of ConvNet, but in this project we will use ResNet as the state-of-the-art in image classification. Residual Networks (ResNet) are developed by Kaiming He et. al. was the winner of ILSVRC in 2015 [4] . They introduced the use of skip connections in order to form a very deep network. Skips connections can be used to ease the training process of such networks, because as the depth of the network increases, the gradients tend to explode or vanish.



Skip connections

Describing the full architecture of ResNet is out of the scope of this study, so I will use them instead, to test on CIFAR-10 dataset. Note that the original architecture is used for ImageNet which has different input size, in this project I will reproduce the model to fit my data.

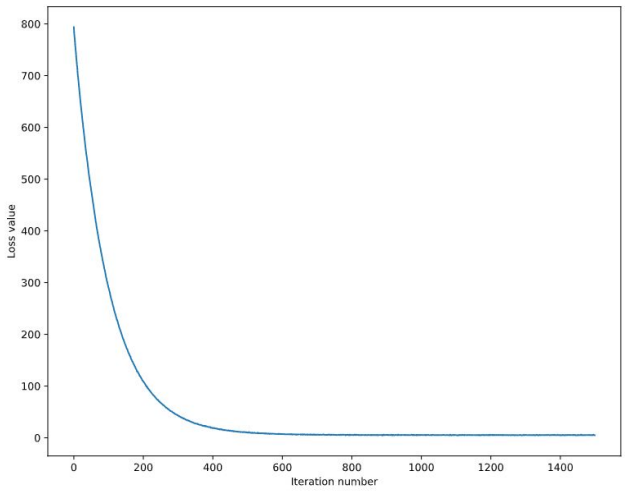
Test on CIFAR-10

Model settings

Now we have the tools and data, we will train our model and test their accuracy.

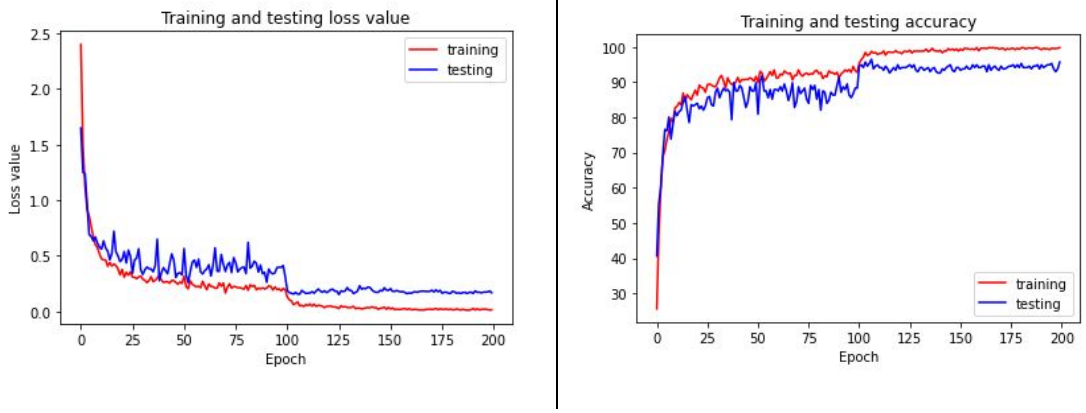
SVM

For SVM model I will use the learning rate of 10^{-7}, regularization of $2.5 \cdot 10^{-4}$ and train for 1500 iterations. Finally, I test this model on 1000 unseen images and get the accuracy of 37.5%. It is quite a bad result but this is a linear model, and with no extra step on extracting the feature from the input, the result is reasonable. We could do better by using some handcraft image processing techniques like histogram of oriented gradients (HOG), principal component analysis (PCA), t-distributed stochastic neighbor embedding (t-SNE) or locally linear embedding (LLE), etc. However, these features extraction techniques are very hard to implement and their performance may not be good compared to modern neural networks. The figure below shows the loss function value of the SVM model.



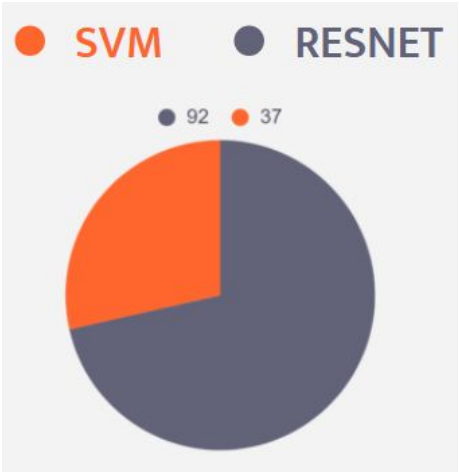
ResNet

There are many variations of ResNet architectures but I will use ResNet50 in this case. For the model implementation, I followed a github page which uses Pytorch framework. I trained the model for 200 epochs with the learning rate of 0.1 and decreased by a factor of 10 if the validation error does not improve for 10 epochs. Stochastic Gradient Descent was employed during the minimization of binary cross entropy loss function with momentum of 0.9 and weight decay of 10^{-4}. The loss function value and accuracy during training and testing of ResNet are shown in the below figure.



Results

As we can see, ResNetl significantly outperforms SVM by 55% in accuracy, proving that neural networks can do a better job in image classification compared to classical image processing methods.

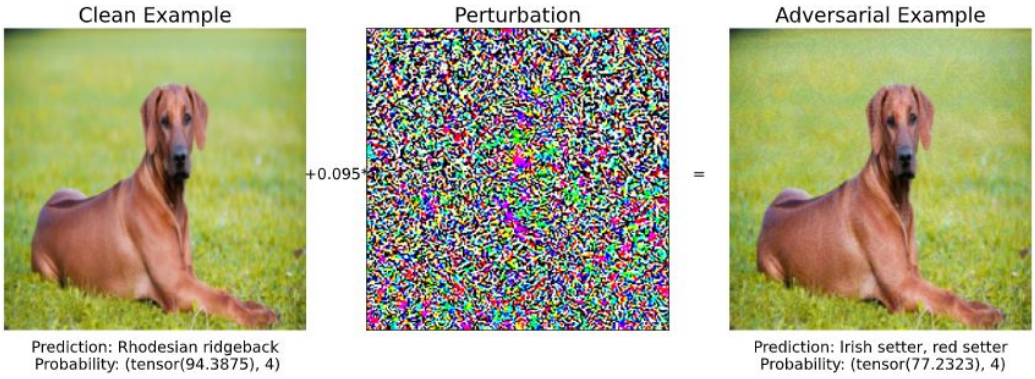


Fooling Neural Networks

Creating adversarial examples

We have seen the power of neural networks so far, which clearly outperform classical image processing methods by a large margin. But until now, we have not known why neural networks work so well. We only treat them as a black box and just apply it without any formal explanation in almost every area, ranging from image classification to natural language processing. That leads to a serious problem in security and consistency. Such problems arise when Szegedy et. al. [1] introduced adversarial examples in 2013. By adding a small amount of noise to the input images, they created images that are unrecognisable to human perception, but completely fool almost state-of-the-art neural networks. This raises a question that could we realize on these architectures ?

If we blindly treat deep learning models as a black box, we may pay the price. For example if an autonomous car instead of recognizing people as people, they recognize them as green light, then the whole system is compromised causing hazards. In the figure below, I illustrate a typical adversarial attack by adding perturbations into a prediction of the ResNet50 model.



As we can see, the model is very confident my image is a ridgeback dog with an accuracy over 90%. However when I add some perturbations, our network completely predicts the wrong class with reasonable accuracy. Interestingly, as a human we cannot recognize what is the difference between those two images but the neural network “sees” them as a different object.

Adversarial examples are not created by adding random noise to the image, these perturbations however are very carefully constructed by using the sign of gradient of the loss function with respect to the inputs. One of the most popular adversarial attacks is Fast Gradient Sign Attack (FGSM) by Goodfellow et. al. [7], which leverages the way neural networks learn to maximize the loss function by adjusting the back-propagated gradients. We can formulate FGSM adversarial attack as follow:

$$x' = x + \varepsilon \cdot sign(\nabla_x J(\theta, x, y))$$

where ε is the perturbation we add to the inputs without causing misclassification to human perception. To do so, ε must be small and follow the sign of the gradient w.r.t. x or in the opposite direction toward the local minimum of the loss function. Note that we also want the input in the range of [0,1] after adding ε.

However, FGSM is still not a powerful attack because it only takes one step to the direction maximizing the loss function. One possible alternative is Projected Gradient Descent (PGD) attack or sometimes called Iterative Fast Gradient Descent. The idea here is almost the same as FGSM but instead of taking a single step in the direction of gradients, they use multiple small steps toward this direction in order to generate adversarial examples. Formally PGD can be represented as follow:

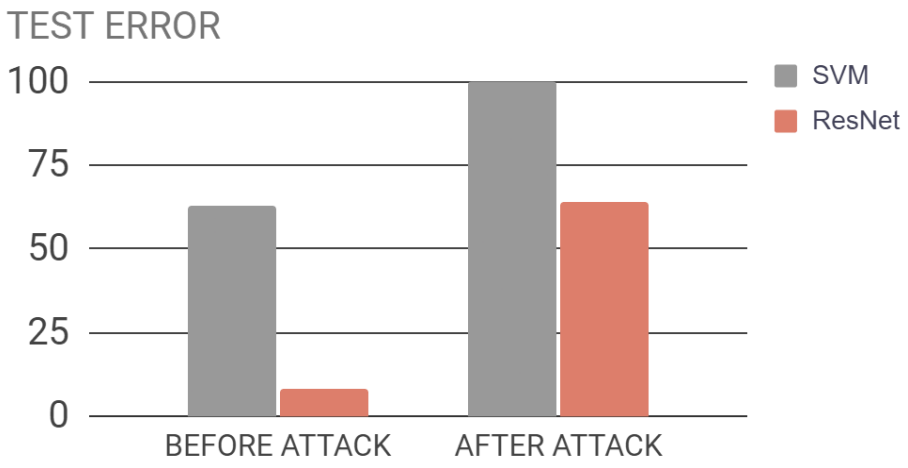
Projected Gradient Descent

$x_0^{adv} = x$
$\alpha = \frac{\varepsilon}{T}$
for t from 0 to T **do**
1. $x_t^{adv} = x_{t-1}^{adv} + \alpha \cdot sign(\nabla_x J(\theta, x_{t-1}^{adv}, y))$

end for

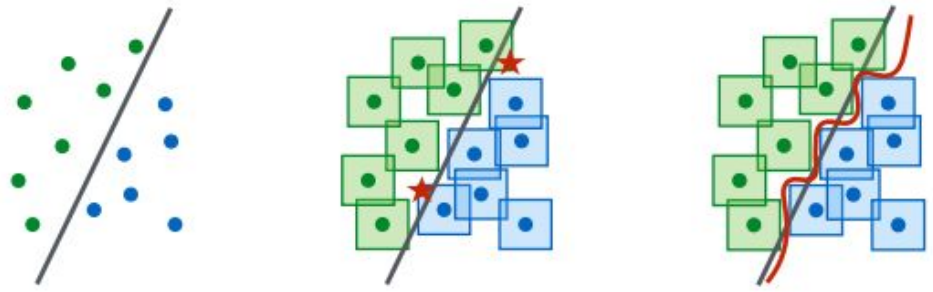
Attacking models

Now we will use PGD to attack both SVM and ResNet models. The below figure shows their error on the test set before and after the attack.



We could clearly see that two of our model's test errors rose up significantly after the PGD attack, SVM model could not correctly predict one single example, and ResNet accuracy decreased from 92% to roughly 30%. Although ResNet's test error was high, it still has some resistance against adversarial examples. At a high level, we can think that adversarial examples

change the decision boundary of the problem to a more complicated one. Thus, it affects more on linear classifiers like SVM. See the below figure for an illustration of standard versus adversarial decision boundary.



Left: a set of points that can be separated by a linear classifier. **Middle:** adversarial decision boundary, where we randomly sample around the data points. Here the red stars indicating the adversarial examples will be misclassified by linear classifiers. **Right:** separate the adversarial examples with a more complicated model, like neural networks.

Robustness against adversarial examples

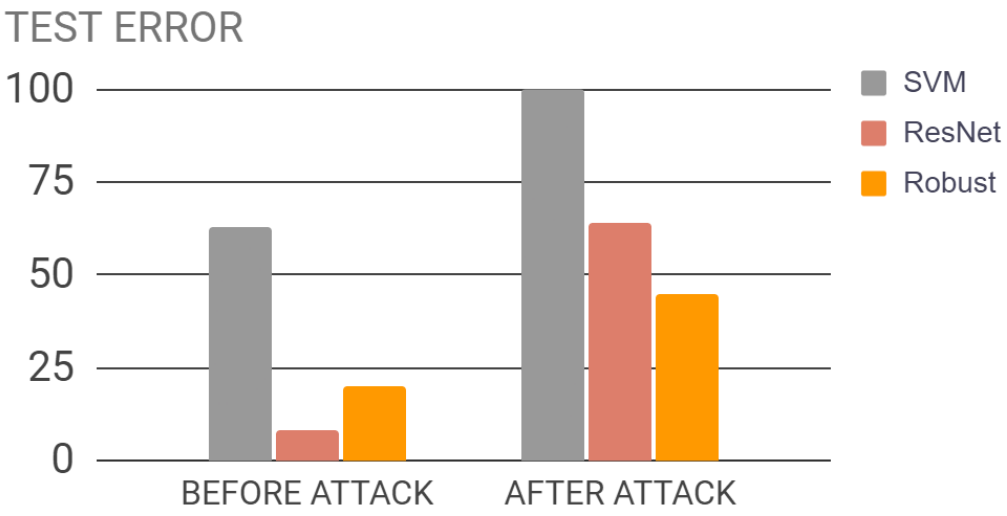
So far we have seen the effect of adversarial examples on neural networks and other linear classifiers. To overcome this problem, we could use defense techniques against adversarial attacks. The basic intuition behind them is using adversarial examples as "ultimate" data augmentation strategy which creates the most confusing examples for the models, and hence increases their robustness. We call these approaches are adversarial training.

In this project I will use a simple strategy, hopefully could increase our ResNet accuracy against adversarial examples. Generating adversarial examples requires the task of maximizing the loss function, adversarial training on the other hand, will minimize the adversarial loss. Formally we can form the this problem mathematically:

$$min_{\theta} \sum_{x,y \in S} max_{\delta \in \Delta} Loss(x + \delta, y, \theta)$$

where S is our training data set, δ is the perturbation we sample from Δ set of points around our sample points in which cannot be differentiated by human perception.

Basically, we train our model normally, however instead of using normal training data, we generate the corresponding adversarial example and feed them to our model. In this experiment, I try adversarial training on the ResNet model only because it is useless to do that on linear classifiers. The result is shown in the figure below. As a matter of fact, when we increase the robustness of a model, their accuracy against clean examples decreases a bit, which is carefully discussed in [8, 9].



From the chart, we can see that Robust ResNet test error is about 45% which is much lower than normal ResNet when encountered with adversarial examples. However, its accuracy on clean examples is declining a bit. I believe that there are more sophisticated methods to train robust models out there.

Conclusion

In this report, firstly, I solved the task of multiclass image classification on the CIFAR-10 dataset. I used Multiclass Support Vector Machine and a novel neural networks architecture, ResNet, to tackle this problem. The results showed that neural networks clearly outperform SVM by a large margin.

Although neural networks are very powerful, they can be fooled easily by adversarial examples, causing misclassification in the images which are unrecognizable by human perception compared with normal data. Therefore, I performed adversarial attacks on the above models, resulting in lower their performance significantly.

Finally, I trained the ResNet model on adversarial examples to increase its robustness against adversarial attacks.

References

[1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna,Dumitru Erhan, Ian Goodfellow, Rob Fergus. Intriguing properties of neural networks. In International Conference on Learning Representations (ICLR), 2013.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[3] Ben-Hur, Asa; Horn, David; Siegelmann, Hava; Vapnik, Vladimir N. ""Support vector clustering" (2001);". Journal of Machine Learning Research. 2: 125–137.

[4] Alex Krizhevsky. Learning multiple layers of features from tiny images. In Technical report, 2009

[5] Nair, Vinod; Hinton, Geoffrey E. (2010), "Rectified Linear Units Improve Restricted Boltzmann Machines", 27th International Conference on International Conference on Machine Learning, ICML'10, USA: Omnipress, pp. 807–814

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In International Conference on Learning Representations (ICLR), 2015.

[7] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In International Conference on Learning Representations, 2017.

[8] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, Aleksander M ¸adry. Robustness May Be at Odds with Accuracy. In International Conference on Learning Representations, 2018.

[9] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In International Conference on Learning Representations, 2018.