Escuela Técnica Superior de Ingenieros Informáticos

Universidad Politécnica de Madrid

Detection of Misclassified and Adversarial examples in Deep Learning

Masther Theses

Master of Science in Artificial Intelligence

AUTHOR: Jorge Marco Blanco
SUPERVISOR: Luis Baumela Molina

2019

Abstract

Great developments have been carried out thanks to deep learning in recent years. However, for a massive adoption in real-world applications, deep learning models need to be trustworthy. In this work we identify scenarios where such models behave in an unexpected fashion, implement a method based on the entropy for assessing their reliability and build a deep learning-based system able to detect its own wrong predictions. Additionally, by using the mentioned method, we achieve a relevant improvement of accuracy with an ensemble of learners.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The need for trustworthy deep learning models

Deep learning techniques have been successfully applied to several fields such as computer vision [Krizhevsky *et al.*, 2012], speech recognition [Amodei *et al.*, 2016-bis], natural language processing [Collobert & Weston, 2008], medicine [Ramsundar *et al.*, 2016], or Go [Silver *et al.*, 2016]. Besides these exceptional progresses, machine learning models should be able to recognize their own ignorance when they encounter a problem they do not know how to solve, like natural intelligence does. Otherwise, wrong decisions would be frequently taken. [Amodei *et al.*, 2016] point out that, in general, machine learning systems may not recognize such a situation, or even worse, they will often assume they are performing well. This general problem of artificial intelligence systems is present as well in deep learning models. For example, [Hendrycks and Gimpel, 2016] report state of the art object recognition classifiers that provide theoretically high reliable predictions when simple noise is introduced as an input. Other authors point out lack of calibration [Guo *et al.*, 2017] and overconfidence in these models [Lee *et al.*, 2017], [Lakshminarayanan *et al.*, 2017]. The absence of uncertainty measurements is indeed a fact in current artificial neural networks architectures [Amodei *et al.*, 2016], , [Gal, 2016]. Evaluating the quality of a neural network prediction is a difficult problem since there does not exist a ground truth for uncertainty that can be used to train in predicting confidence.

In summary, lack of robustness, confidence or uncertainty estimators are problems that deep learning models need to overcome in order to consolidate its presence in real-world applications. As a matter of fact, the requirements for artificial intelligence systems are becoming harder. See, for instance: [European Comission, 2019] (bold text original from the technical report)

> *The European AI strategy and the coordinated plan make clear that* ***trust is a prerequisite to ensure a human-centric approach to AI****: AI is not an end in itself, but a tool that has to serve people with the ultimate aim of increasing human well-being. To achieve this, the*

> ***trustworthiness of AI should be ensured.*** *The values on which our societies are based need to be fully integrated in the way AI develops.*

Although the concept of 'trustworthy models' may have subjective or ambiguous meanings, [Lipton, 2016] sheds some light on this topic. He points out the attributes of accuracy and foreseeable performance, such that behave as expected with respect to its real objectives and scenarios, for deserving such a qualification. Our point of view is that both are, if not sufficient, at least necessary attributes for building trustworthy models.

1.1.1 Contribution

Coherently with this necessity, in this work we develop tools and methods that improve the trustworthiness of deep neural networks models. Specifically, we will focus on the topic deep learning for supervised classification.

First of all, we derive from basic intuitions a measure based on the Kullback-Leiber divergence and the entropy that allows evaluating the confidence in the models predictions. After testing it with some preliminary experiments, we make use of it to build a system based on a neural network model, whose accuracy can be specified beforehand. Though, the cost of gaining additional accuracy in predictions is the need to perform manual revisions in some of them, in such a way that if a high degree of accuracy is required, the system will reject more predictions which will be derived for manual evaluation. On the contrary, if more automation is required, the system will 'pass' more predictions, even tough some of them are actually wrongly classified, although always in a lower or equal rate to the original model error. Furthermore, the system possesses a new skill that the single neural network classifier does not, the ability to detect adversarial and out of range examples. In consequence, the system features some of the required attributes for trustworthiness, i.e. accuracy and performance.

1.2 Supervised classification with neural networks

Supervised learning is the process of building a function that maps an input to an output based on previous examples. When the process of learning is done by a neural network, it can be defined [Haykin, 2009] as:

> *(...) supervised learning involves the modification of the synaptic weights of a neural network by applying a set of labeled training examples, or task examples. Each example consists of a unique input signal and a corresponding desired (target) response. The network is presented with an example picked at random from the set, and the synaptic weights (free parameters) of the network are modified to minimize the difference between the desired response and the actual response of the network produced by the input signal in accordance with an appropriate statistical criterion. The training of the network is repeated for many examples in the set,*

until the network reaches a steady state where there are no further significant changes in the synaptic weights. The previously applied training examples may be reapplied during the training session, but in a different order. Thus the network learns from the examples by constructing an input–output mapping for the problem at hand.

A key part of this process is the training methodology that involves choosing the right set mentioned input-ouput examples, also called datasets. We will frequently use these concepts in this document, therefore we will to do a general introduction to this topic.

1.2.1 Training methodology

As mentioned in the last section, a neural network learns by tuning its parameters (weights) to adapt to known examples in such a way that it is able to deal with unknown examples that might be presented in the future. Such an ability is called generalization.

In order to implement the mentioned methodology, at least two datasets with labeled examples are needed, one that the model uses to learn (training data set) and another to evaluate it, i.e., to measure its generalization ability. An optional third set of examples might be used as well for training hyperparameters. The different kinds of datasets are formally defined by [Ripley, 2017], as:

Training dataset. A set of examples used for learning, that is to fit the parameters (i.e., weights) of the classifier.

Validation or development dataset. A set of examples used to tune the hyperparameters (i.e., architecture, not weights) of a classifier. For example, to choose the number of hidden units in a neural network.

Test dataset. A set of examples used only to assess the performance (generalization) of a fully-specified classifier.

Intuitively, for a correct design of the training process, the 3 sets should belong to the same probability distribution and each one has to be independent from the other two. A formalization of this idea is provided by [Zhou, 2012]:

Denote X as the instance space, D as a distribution over X , and f the ground-truth target function. Given a training data set $D = (x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$, where the instances x_i are drawn i.i.d. (independently and identically distributed) from D and $y_i = f(x_i)$, taking classification as an example, the goal is to construct a learner h which minimizes the generalization error.

1.2.2 Deep neural networks

More layers in a neural network provide stronger abstraction and representational power, thus the capacity to discover features in an independent fashion, with the consequence of an enhanced generalization ability and the capacity to solve more complex tasks. Building and running neural networks with more layers (deeper), has been possible thanks to advances in computational capacity like the development of faster CPUs, or the availability of GPUs, simultaneously with the progress of software for distributed computing. Also, the availability of huge labeled dataset has been necessary. As [Goodfellow *et al.*, 2016] highlight, *'As of 2016, a rough rule of thumbs that a supervised deep learning algorithm will generally achieve acceptable performance with around 5,000 labeled examples per category and will match or exceed human performance when trained with a dataset containing at least 10million labeled examples'*. Finally, the improvement of training algorithms has been the final factor for the resurgence of neural networks.

Nevertheless, as showed in last sections, some problems related to this fast development have arisen simultaneously.

1.3 State of the art

As a critical issue for the massive adoption of deep learning, there is an extensive literature regarding the measure of confidence of deep models. We mention those works that we consider more significant in terms of applicability.

[Gal, 2016] suggests that classifiers should return an output, complementary to the label, providing the level of uncertainty (e.g. in the case of out of distribution examples.). He claims that uncertainty information can be obtained from neural networks by adding a dropout layer [Srivastava *et al.*, 2014], on the basis that the new architecture behaves approximately as a Bayesian neural network. However, some of the authors have later highlighted some doubts about this method reliability [Hron *et al.*, 2016].

[Lee *et al.*, 2017] proposes a method based on a new loss function, called *confidence loss*, to train neural networks that would allow detecting out of distribution examples. [Hafner *et al.*, 2018] use recently developed implementations of Bayesian Neural networks to achieve reliable uncertainty estimates, specifically using the so-called *noise contrastive priors*. Though promising, the need to re-train existing networks might be a drawback.

[Guo *et al.*, 2017] define model calibration in terms of the likelihood of its predictions and show that modern deep learning networks are poorly calibrated compared to older models built with less number of parameters. The authors propose a method by which only by modifying a single parameter called *temperature scaling*, calibration is improved without affecting the accuracy.

[Lakshminarayanan *et al.*, 2017] use ensembles of learners to average predictions and capture "model uncertainty" as well as adversarial training to smooth predictive estimates. In their work, the entropy of the predictive distributions of ensembles is used to compare the uncertainty estimates of different models. [Hendrycks and Gimpel, 2016] present a baseline for detecting error and out-of-distribution examples derived directly from softmax distributions and propose a set of standard methodology and metrics based on the statistics derived from the output distributions for assessing the automatic detection of these examples.

Although several lines of research are opened, we are mainly inspired by the methods and ideas from the last two cited works, besides their relevance, because they can be applied to existing networks. In our opinion, that is a very important advantage given the existing capital in trained networks, created by means of computational investment.

Chapter 2

Deep learning classifier output analysis

Choosing the right performance metrics to evaluate a classifier is key for the model success. The most common metric is accuracy, that represents the fraction of correct predictions over all the predictions. It is defined as

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of total predictions}} \tag{2.1}$$

Accuracy is a very descriptive metric, however, it may provide a false feeling of achieving high performance under some circumstances. When it comes to problems with imbalanced classes distributions (i.e. detection of fraudulent transactions that represents a very small percentage of the population, diseases detection..), additional metrics should be considered in order to avoid misleading conclusions from the measurement of accuracy. A popular and complementary magnitude to accuracy is the loss function. Unlike accuracy, loss is not a percentage, but the sum of the errors made for each example in training or validation sets. It can be calculated on training and validation datasets, and may also help to identify overfitting. There are several variants of the loss function, being the most frequent the cross-entropy loss or log loss [Bishop, 2006]. It can be expressed as:

$$\text{Cross entropy loss} = \frac{-1}{N} \sum_{i}^{N} \sum_{j}^{N} y_{ij} log(p_{ij}) \tag{2.2}$$

where y_{ij} indicates whether the example i belongs to class j and p_{ij} tells us the probability of example i belonging to class j, as predicted by the model.

Additionally, other magnitudes such as confusion matrices can be used for a better understanding of the model results. In all cases, these metrics are based exclusively on choosing a ‘winner’ with the highest score. Scores assigned to ’loser’ classes are neglected, but not surprisingly, these scores contain relevant information, as we will show later. Before that, we will introduce the softmax function, the most common method of representing the output distribution of a supervised classifier in deep learning.

2.1 Softmax function

In the problem of supervised classification, the labels representing the classes are encoded as one hot vectors. For instance, in a problem for the recognition of 10 kinds of objects, the airplane could be encoded as $\boldsymbol{Y_0} = \{1, 0, ..., 0\}$, the car as $\boldsymbol{Y_1} = \{0, 1, 0, ..., 0\}$ and so on. From these examples, we observe that one hot vectors features the properties of a probability density function, thus if the predictive distribution (or output vectors) from our model behave as a probability density function as well, comparing model predictions with the ground truth will be much easier. Probably for this reason, the standard in neural networks design is to place a function that normalizes the scores to a probability density function, as the last layer of the network. This function is called softmax. It is defined in the range $S : R^d \to R^d$ where d is the number of classes, and is expressed as:

$$S(\mathbf{z})_i = \frac{e^{-\beta z_i}}{\sum_{j=1}^{d} e^{-\beta z_j}} \tag{2.3}$$

It is also very common to assign to β the value 1, resulting in a simplified softmax function:

$$S(\mathbf{z})_i = \frac{e^{-z_i}}{\sum_{j=1}^{d} e^{-z_j}} \tag{2.4}$$

Note that no matter what value is assigned to β, we will get a probability density function where the classes will be sorted on the same way by their ‘probabilities’.

Intuitively, there should exist a relation among the shape of the output function and the confidence the network has in its prediction. The output density function should resemble to a one hot vector(a sharper form), when the network is more confident about the prediction, and smoother when the prediction is less accurate. Under this assumption, given the properties that the softmax function confers to the output distribution, the individual scores given to each class can be interpreted as probabilities. In this sense, we could be very confident in a prediction if the winner class obtains a probability (score) of 99% , and doubtful if it obtains a probability of only 51%. Note there are no theoretical grounds for this assumption, neither experiments that support it. In a test conducted with a state of the art network trained on the cifar-10 dataset, [Hendrycks and Gimpel, 2016] report accuracies of over 90% on images that are just noise. We have also confirmed and extended these results carrying out the following experiment. 100 noisy images inputs (see figure 2.1) have been introduced in a convolutional network trained on the MNIST dataset. While a uniform density function in the labels predicted should be expected, results show a relevant bias of the network towards certain labels. Additionally, over 10% of the winner classes have a ‘probability’ of over 0.7 (see figure 2.2).

Given these results, an alternative approach is needed to provide a measure of confidence. We might think of a distance or difference between the output and the actual label (recall truth labels are encoded as one hot vectors), but the problem

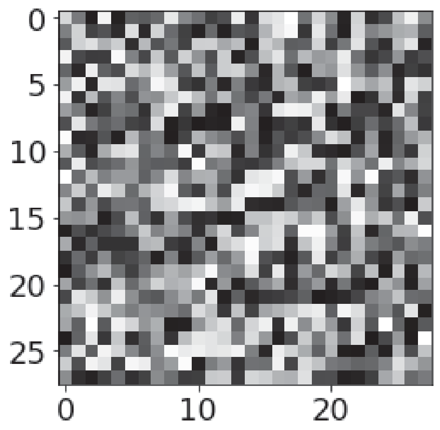


Figure 2.1: Sample noisy input.

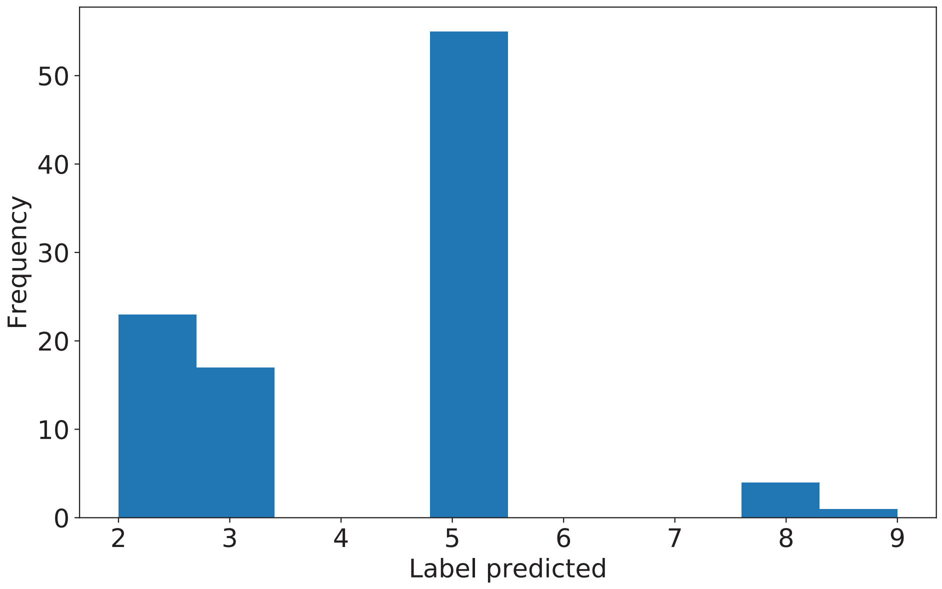


Figure 2.2: Convolutional network trained on **Mnist** dataset. Labels predicted on 100 randomly generated noisy inputs.

is obvious: in test or production scenarios we do not know the actual label, so we cannot measure any distance to it. However, we can take this argument from the opposite side and measure the distance of the output distribution, not to a one hot vector, but to its contrary, the uniform distribution. The posed hypothesis is that the larger the distance to the uniform distribution, the shorter the distance to a one hot vector, thus the stronger the prediction. This point of view is coherent with the rules for knowledge representation inside an artificial neural network described by [Haykin, 2009]:

Rule 1 Similar inputs (i.e. patterns drawn) from similar classes should usually produce similar representations inside the network, and should therefore be classified as belonging to the same class. (...)

Rule 2 Items to be categorized as separate classes should be given widely different representations in the networks.

A natural choice for implementing the distance is the L_2 norm in R^d where d is the number of classes. Unfortunately, as [Aggarwal *et al.*, 2001] and [Beyer *et al.*, 2014] highlight, the L2 norm doesn't behave as expected in high dimensions spaces, literally:

Under certain reasonable assumptions on the data distribution, the ratio of the distances of the nearest and farthest neighbors to a given target in high dimensional space is almost 1 for a wide variety of data distributions and distance functions. In such a case, the nearest neighbor problem becomes ill defined, since the contrast between the distances to different data points does not exist.

[Aggarwal *et al.*, 2001] suggest to use the L_1 norm or other fractional distance metric L_k where k is a fraction smaller than one. However, we will attempt an alternative approach based on the theory of information that might be more intuitive. The Kullback-Leibler divergence is a common magnitude for comparing distributions. As we will see, it is not a symmetric function and thus cannot be considered as a distance. Nonetheless, since our goal is to compare different output distributions against one fixed distribution (the uniform distribution), it might be useful for our purpose. Before describing the Kullback-Leibler divergence we will recall the entropy, the basic magnitude in the theory of information and an antecedent, the entropy in statistical mechanics.

2.2 Output analysis with the theory of information

2.2.1 Origin: Entropy in statistical mechanics

Ludwig Boltzmann developed the concept of entropy to explain the behavior of macroscopic systems in terms of the physical laws of the microscopic components. It is defined as the natural logarithm of the number of possible states Ω that the particles inside a gas can reach, multiplied by a constant.

$$S = k_b \log(\Omega).$$

Thanks to thermodynamics, we know that systems of particles evolve to states in which the arrangement of the particles is more probable, eventually reaching the equilibrium, that is the state with the highest probability.

2.2.2 Entropy in Theory of information: Shannon entropy

Entropy in the theory of information was defined by Claude E. Shannon in 1948 with a very similar equation [Shannon 1948] as:

$$H = -\sum_i p_i \log_b p_i. \tag{2.5}$$

It may be interpreted as the amount of information transmitted in a message and since it was used to analyze binary coded information, base 2 logarithm was used. In this work we will use the natural logarithm.

2.2.3 Kullback-Leibler divergence

Given two probability density functions p and q, defined in the sample space $S = 1, 2, ..., N$, the Kullback-Leibler divergence is defined as:

$$D(p||q) = \sum_{x=1}^{N} p(x) \log \frac{p(x)}{q(x)}.$$

If we consider p as the actual function from which we are sampling, and q the estimated distribution for p, we can think of the Kullback-Leibler divergence as the average amount of (extra) information we receive if we sampled from p in stead of q.

Our goal is to measure the Kullback-Leibler divergence of our output function P to an uniform density function:

$$\begin{gathered}
KL(P||uniform) = \sum_{x=1}^{N} p(x) \log \frac{p(x)}{q(uniform)} = \\
\sum_{x=1}^{N} p(x) \log \frac{p(x)}{q(1/N)} = \sum_{x=1}^{N} p(x) \log p(x) - \sum_{x=1}^{N} p(x) \log(1/N) = \\
\sum_{x=1}^{N} p(x) \log p(x) - N \log(1/N) \sum_{x=1}^{N} p(x) = \\
\sum_{x=1}^{N} p(x) \log p(x) + Nlog(N),
\end{gathered} \tag{2.6}$$

where we have applied $\sum_{x=1}^{N} p(x) = 1$ since P is a probability density function. The first term of the equation

$$\sum_{x=1}^{N} p(x) \log p(x),$$

is simply the formula of the entropy 2.5 using the natural logarithm multiplied by (-1). Therefore, we obtain:

$$KL(P||uniform) = Nlog(N) - H(P). \tag{2.7}$$

Since $p(x) \leq 1$ for any value of x, given the definition of a probability density function, and $N \geq 1$ given the description of the problem, we conclude that $H(P)$ is non-negative and that $KL(P||uniform)$ decreases as $H(P)$ increases.

2.3 Conclusions and preliminary observations

As per our hypothesis, the more confident predictive distributions are those more different from the uniform distribution. From 2.7, they are simply those with a lower entropy H. In order to test this conclusion, we conduct the following experiment. With the same network used to compute the predictions of figure 2.2, the entropies of the output functions corresponding to 100 noisy inputs are compared with the entropies of the outputs corresponding 100 images of the MNIST test dataset. The results are shown in figure 2.3. By observing the figure, we could differentiate the examples belonging to the dataset distribution from the noisy images. Even more, we could set a frontier among them.

This result is coherent with the hypothesis and thus encourages us to continue walking this way. In the next section, we will present a method that takes advantage of these results.

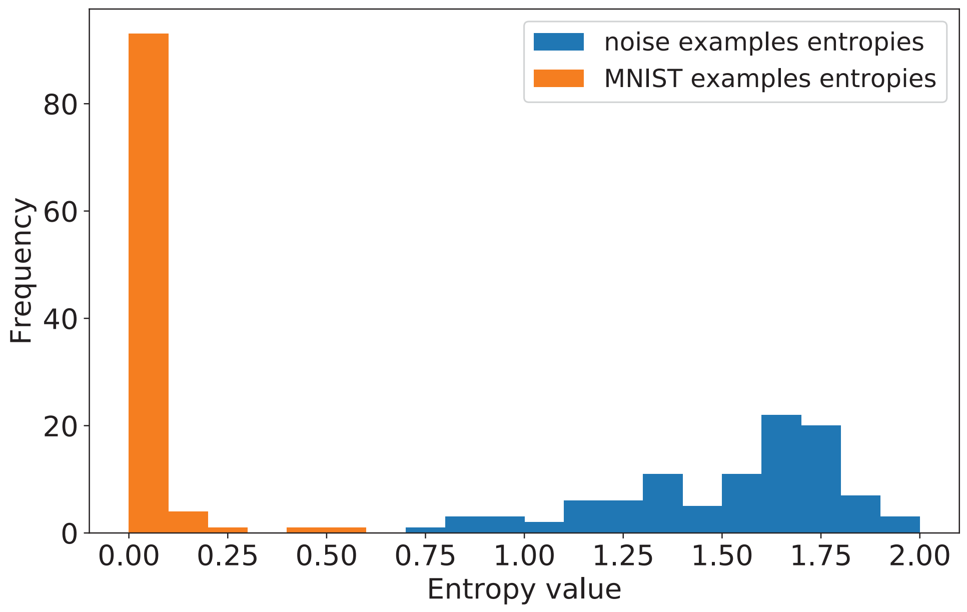


Figure 2.3: Histogram of entropy frequencies of output functions corresponding for in range and out of range inputs. The β parameter of the softmax function is 1.

Chapter 3

A method to set confidence working ranges in deep models

The presented method consists of adding a new stage to a deep learning classifier. This new stage is a binary classifier that predicts whether the network prediction is right or wrong based on the entropy of the output distribution. As concluded in the last section and seen in figure 2.3, we know that the lower the entropy, the more confident is the prediction, but we do not have a way to set an exact border among right and wrong predictions. What we do is to arbitrarily set a threshold below which the predictions are considered wrong and above which are considered right. Predictions whose entropy is above the threshold are not accepted and thus need a manual revision for correct classification, while predictions whose entropy is below the threshold are considered confident enough, thus the system can continue with its normal operations in an automated way. If a given system requires to work as much as possible in an automated fashion, we set a higher threshold so that more predictions will 'pass'. The trade off is that the higher the threshold, the more wrong predictions will 'pass'.

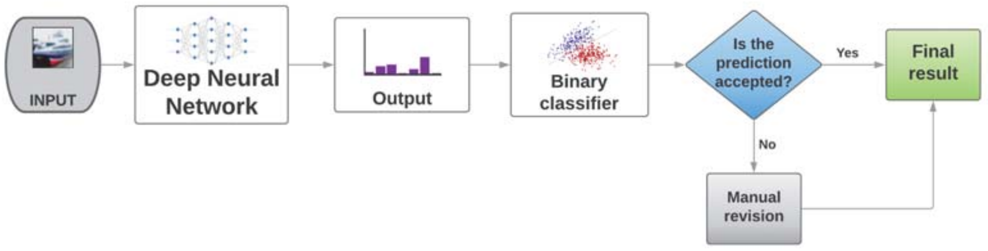


Figure 3.1: System for detecting classification errors

On the contrary, if the allowance for wrong predictions is very low (in scenarios such as cancer detection or autonomous vehicles), we will set a threshold with a low value. In this case, we can be very confident in the predictions classified as right, but other right predictions will not 'pass' the threshold and would require an unneeded manual supervision. As an extreme use case example, let us suppose we have a deep model with a 91% accuracy. If the binary classifier, is set to work in an absolute automatic way, all the predictions will be accepted, but 9% of the predictions will be wrong.

An important feature of the described methodology is that it can be applied directly to existing models, just adding the second stage after the output distribution to evaluate it. In this way, there is no need to carry out computational expensive re-trainings.

3.1 Method implementation

Based on these intuitions and on the evidences from the experiments represented in figure 2.3, we build a framework to implement the method. Once the classifier we want to enhance is trained with the training set (See 1.2.1), we study the entropies of its predictions on the whole validation set. The entropy threshold will be shifted from the minimum value of the entropy (0) to its maximum ($-ln(\frac{1}{d})$), where d is the number of classes) in a discrete number of equal length steps. For each value of the threshold, all the predictions are evaluated. Those whose entropy is located to the left of the threshold (lower entropy) are considered as correct predictions, while those located to the right are considered as wrong. To better analyze these results, we build a confusion matrix for each step of the threshold as follows:

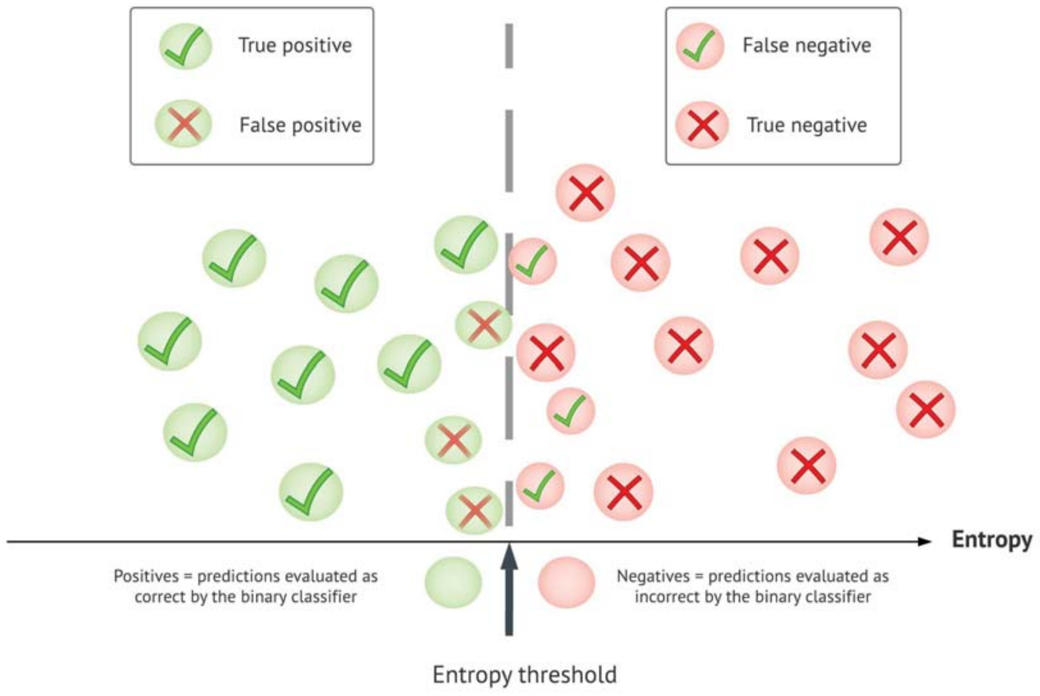

True positive
False positive
False negative
True negative
Entropy
Positives = predictions evaluated as correct by the binary classifier
Negatives = predictions evaluated as incorrect by the binary classifier
Entropy threshold

Figure 3.2: Entropy values and classifier predictions.

3.1.1 Confusion matrix

A confusion matrix for a binary classifier [Townsend, 1971] is a matrix of 2x2 dimension. Its four elements are named true positives, false positives, true negatives and false negatives. In our specific case, we define the elements as follows:

True Positives (TP) Examples correctly classified by the model and identified as correct by the binary classifier.

False Positives (FP) Examples wrongly classified by the model, but evaluated as correct by the binary classifier.

True Negatives (TN) Examples wrongly classified by the model and identified as incorrect by the binary classifier.

False Negatives (FN) Examples correctly classified by the model, but evaluated as incorrect by the binary classifier.

Putting together the metrics of the confusion matrix for each step, we can build a precision recall curve to gain an overview of the performance of the system.

3.1.2 Precision recall

The components of a precision recall curve are:

Precision. Tells us the proportion of true positives among all predicted positives:

$$\text{Precision} = \frac{TP}{TP + FP}$$

In our case, we interpret it as an enhanced accuracy since it tells us the correct predictions rate of the entire system.

Recall. It is the proportion of true positives among all actual positives:

$$\text{Recall} = \frac{TP}{TP + FN}$$

In our case, it represents the rate of cases in which we will need to unnecessary revise a prediction because it was actually correctly classified.

We will build a curve with the relationship between precision and recall for each value of the entropy threshold. Thanks to this curve, we can select the desired range of precision for our system and it will tell us the price we have to pay for that precision in terms of manual revisions of the results. In this sense, it may be interesting to study the rate of automation of our system, i.e. how often a manual revision will not be necessary.

3.1.3 Automation rate

We define the automation rate of the system as the fraction of predictions for which there will be no manual revision (evaluated as correct by the binary classifier, i.e. positives) over all the predictions.

$$\text{Automation rate} = \frac{TP + FP}{TP + FP + TN + FN}$$

3.2 Conclusions

In the analysis developed in this section we have only dealt with examples from the the training and test data, drawn both from an original distribution D of an instance space X (see section 1.2.1). In this context, we refer to misclassified examples as those belonging to the original distribution D that are incorrectly classified. Unfortunately, there exist examples of a kind that the classifier have never seen before (they are not drawn from the referred distribution D). For instance, in the case of the MNIST dataset, any digit would belong to the original distribution D. On the contrary, characters of the alphabet or any other image not representing a digit cannot come from distribution D and are considered as out of distribution or out of range examples.

There are also images that have been explicitly manipulated to cheat the classifier in such a way that they apparently belong to the distribution D (from the perspective of a human observer) when they actually do not. Those are called adversarial examples and will be examined in the following section.

Chapter 4

Detection of Classification Errors

Trustworthy deep networks for real world applications, if cannot avoid making prediction mistakes, they should have the ability of being conscious of them. As explained in last section, the mistakes may be classified in three groups attending to the nature of the input example that caused it: misclassification, out of range examples and adversarial examples. In this section we will study them separately.

4.1 Types of classification errors

4.1.1 Misclassification

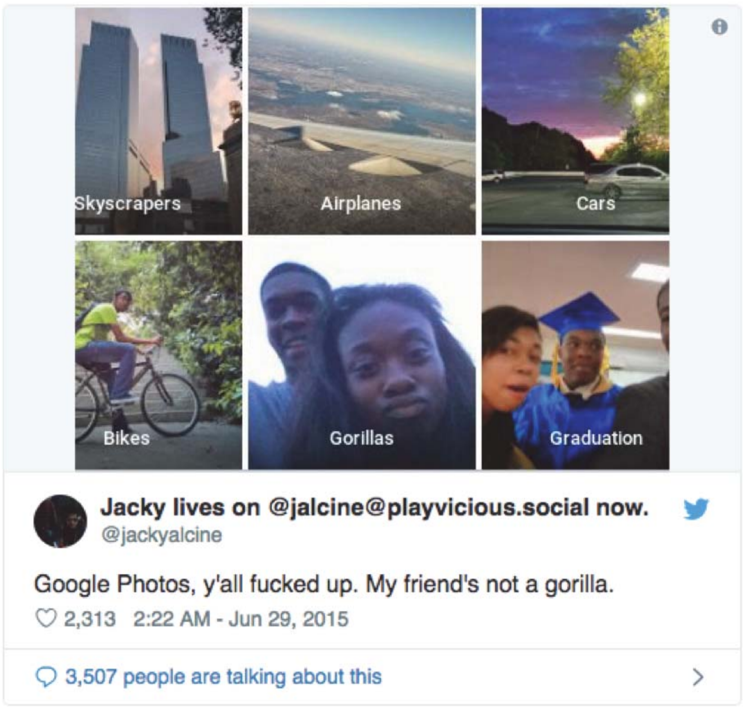
Misclassification refers to a mistake made by the model when labeling an example that belongs to the distribution D (see 1.2.1). As a real example, an error in a Google app for android devices was reported by [Grush, 2015]. An afroamerican was tagged as a gorilla by this app, which caused severe complaints (see 4.1)

However, these are the actually the easy ones, since they can be solved by just improving the accuracy of the model.

4.1.2 Out of distribution examples

Out of distribution examples refers to examples not belonging to the distribution D. By its own definition they are impossible to classify correctly, since every possible output represents a class from D. Also, training a network for detecting them would be a enormous effort. In the case of object recognition, we should train the network with samples of all possible existing objects! Note, however, that as datasets grow with more classes, the complimentary set of out of range objects decreases since the total number of existing objects is finite, although it is a really long way to cover it completely.

These examples are problematic because the classifier will always make a wrong prediction, even with high confidence (as seen in 1.2.1). Since we do not want to wait until a dataset covering all possible objects is available, our ambition is just to detect these examples before a wrong class is predicted.



Jacky lives on @jalcine@playvicious.social now.
@jackyalcine

Google Photos, y'all fucked up. My friend's not a gorilla.

2,313 2:22 AM - Jun 29, 2015

3,507 people are talking about this

Figure 4.1: An example of misclassification mistake.

A real world case of a mistake caused by an out of distribution example has been recently reported by [Dodds, 2018]. A Chinese actress was fined by a facial recognition system designed to catch jaywalkers (see figure 4.2). What the camera actually caught is an advert featuring her face placed on the side of a passing bus. Since the classifier was not trained for detecting photos of people and thus differentiating them of real people, the image was wrongly classified.

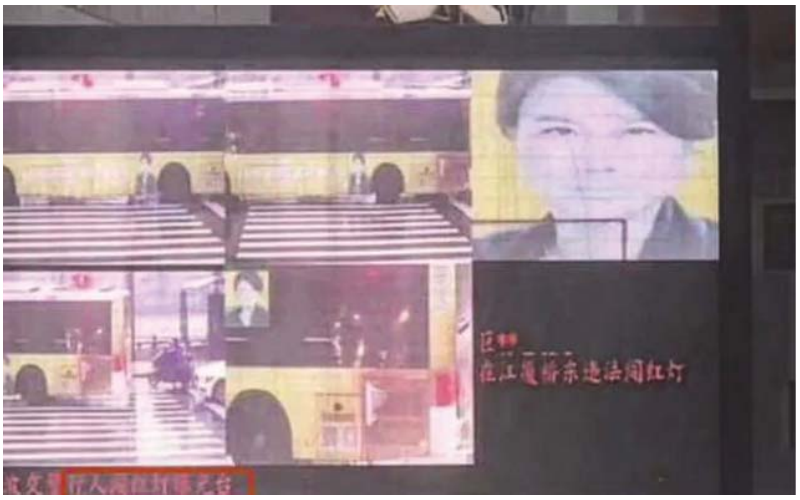


Figure 4.2: Classification Mistake caused by an out of range example.

4.1.3 Adversarial examples

Adversarial examples are defined by [Goodfellow *et al.*, 2014]:

> *Inputs formed by applying small but intentionally worst-case perturbations such that the perturbed input results in the model outputting an incorrect answer with high confidence.*

From this definition, we can interpret adversarial examples in our framework as those that apparently belong to the training distribution (for a human observer), but actually do not (see figure 4.3). There exist methods to train the classifiers against these attacks [Madry *et al.*, 2017], and also methods to circumvent those defenses [Athalye *et al.*, 2018]. However, to our knowledge there is no method designed to detect them.

As a real world example, [Eykholt *et al.*, 2018] explain that it is possible to place a certain black-and-white sticker on a real stop sign, in such a way that the signal recognition model of a driverless car could misclassify the sign, and cause it to not stop.

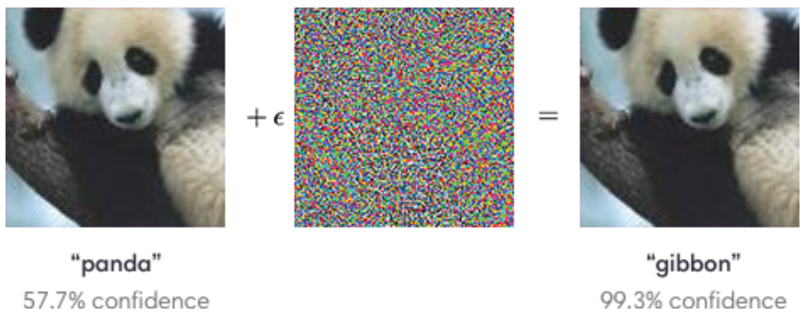


Figure 4.3: A demonstration of adversarial example generation applied to GoogLeNet convolution network on ImageNet. Demonstration from [Goodfellow *et al.*, 2014]

Once we have defined the different kinds of classification mistakes, we will use them to evaluate the system presented in the last section. As we will see, an important degree of detection rates is achieved.

Chapter 5

Accuracy improvement with ensemble learning

Ensemble learning is a technique based on the paradigm of collective learning, that takes the classic idea of combining the knowledge of experts before taking a final decision. It has been widely used in several knowledge fields. For example the jury in an Olympics game contest, the board of directors of a company, or a minister council have used it.

In the field of machine learning, this paradigm has been successfully applied since the 90's. Bagging [Breiman, 1996], Random forests [Breiman, 2001], or XG-Boost [Chen & Guestrin, 2016] are some representative methods. One of the causes of this success might come from The No Free Lunch Theorem [Wolpert, 1996], [Wolpert & Macready, 1997] which implies that it is not possible to obtain a learning algorithm that is consistently better than other learning algorithms in different circumstances. When dealing with large and sparse datasets from which we want to learn, it is reasonable to think that some learners will perform better in some regions, and other learners will do in others.

In this theses, we make use of this paradigm, together with the hypothesis that the lower the entropy of a classifier distribution output, the better the prediction. As a result of this combination, we obtain a significant improvement of the accuracy compared with that obtained from a single learner.

5.1 Stacking

There are several methods based in collective learning to create supervised classifiers with improved performance. The common procedure is to train a set of models under different circumstances, so that every individual specializes in diverse angles of the problem. Afterwards, their opinions are combined in order to widen the knowledge of the group. A key feature for the success of an ensemble is the diversity among the classifiers [Zhou, 2012], [Tumer & Ghosh, 1995], which, as a general rule, reduces the variance.

Another successful procedure of collective learning is stacking, created by

[Wolpert, 1992]. It is based on a parallel architecture of so-called 'first-level' learners and a combiner called 'second-level' learner or meta-learner. Under this paradigm, the first-level learners are trained with the training dataset and produce a combined output that plays the role of a new training dataset. This new dataset is used for training the second-level learner.

5.2 Accuracy improvement with a stacking architecture

In this work, we build an ensemble of neural networks with the stacking architecture. The networks have the same structure and they are all trained on the same dataset. The difference among them is the initialization of the weights, which are randomly drawn from a probability distribution and provide the system with the desired diversity. [Lakshminarayanan *et al.*, 2017] have built a similar ensemble architecture and found that random initialization and shuffling of the data points is enough for a good performance, and that applying a bagging strategy deteriorated it. In our case, the task of the second-level learner is much easier than the general case described in the previous section. After analyzing the single predictions, it will choose the one with a lower entropy as it is expected to be the most confident as per our findings. A representation of this schema is shown in figure 5.1. Using this method, we obtain a significant improvement of accuracy and in a more efficient way, in terms of network size, than other existing methods.

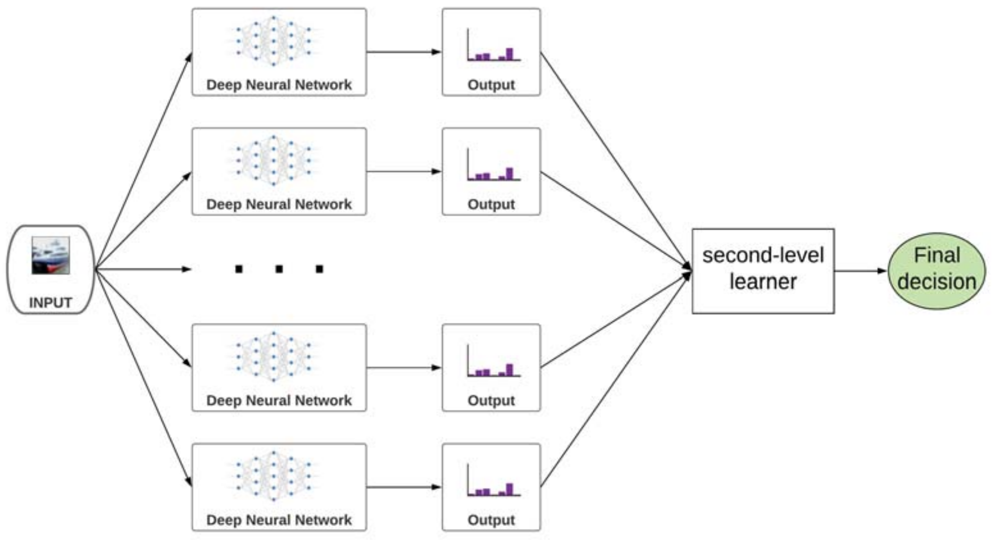


Figure 5.1: Schema of the ensemble of neural networks. Every network has the same structure, but a different initialization of the weights.

Chapter 6

Experiments

For conducting our experiments we will use ResNet(Residual Network) architecture [He *et al.*, 2016]. This is a very successful deep learning architecture that solves the problem of accuracy saturation in 'deeper' learning models and has become a standard in the field. The selected datasets for carrying out the experiments are CIFAR-10 [Krizhevsky *et al.*, 2009] and Fashion-MNIST [Xiao *et al.*, 2017] datasets. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. Fashion-MNIST is a dataset of fashion article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

6.1 System implementation

We train the ResNet v20 version of ResNet architecture composed of 20 layers and 270K parameters on both datasets. We achieve an initial accuracy of 0.916 on CIFAR10 (similar to the accuracy obtained in the original paper) and 0.9237 on Fashion-MNIST dataset. The networks are employed to implement the methodology described in section 3 as follows. First of all, the curve is built calculating precision and recall points for different entropy-threshold values. Afterwards, the detection ability of the system in out of range and adversarial examples is evaluated for a set points in the curve.

6.1.1 Adversarial detection

Adversarial examples used in the experiment have been generated using the fast gradient method described in [Goodfellow *et al.*, 2014] by means of the keras implementation developed by IBM in [Nicolae, 2018]. We have selected an attack strength of $\epsilon = 0.02$ that causes the accuracy to drop from 0.916 to 0.298 in the network trained on CIFAR-10 dataset, and from 0.924 to 0.414 in the network trained on fashion-MNIST, while images do not experiment many noticeable changes for a human observer (see figures 6.1 and 6.2)

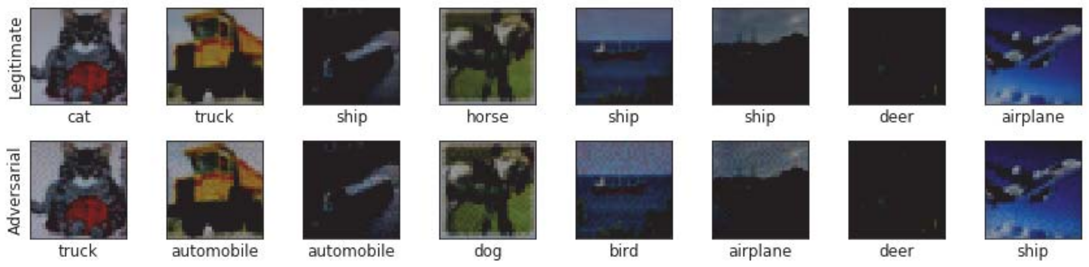


Figure 6.1: Legitimate and adversarial examples from CIFAR-10 dataset used in the experiment. Predicted labels are shown below each image.

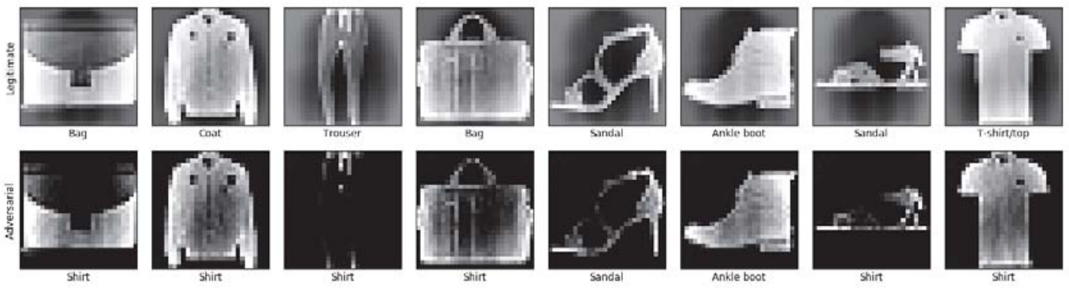


Figure 6.2: Legitimate and adversarial examples from Fashion-MNIST dataset used in the experiment. Predicted labels are shown below each image.

The detection rate of adversarial examples has been calculated over those examples that succeed in cheating the network:

$$\text{Adversarial detection rate} = \frac{\text{number of detected attacks}}{\text{number of successful attacks}}.$$

Alternatively, we could have used the number of attacks in the denominator for calculating the adversarial detection rate. If we used this magnitude, for the case of a very weak attack (small ϵ) the detection rate would be very low, what wouldn't reflect the actual system defensive performance, since almost no harm would have been produced to the system (a very weak attack is not able to cheat a classifier).

6.1.2 Out of range detection

Out of range examples for the experiments are created as images of the same size as those from the corresponding dataset, whose pixel values are randomly generated.

6.1.3 Results

Precision recall curves obtained for both datasets are shown in figures 6.3 and 6.4. These curves are used to build the different working scenarios shown in tables 6.1 and 6.2.

Taking into account the concepts learned in section 3, we interpret the results of a row from table 6.1 as follows: If a precision of 0.98 is required, we will need to perform a 19% of manual revisions among all our predictions, a 85% of the correct predictions were accepted by the binary classifiers (the other 15% will be superfluously revised since they are rejected and will pass to manual revision). Additionally, under this working specification, the system achieves an adversarial examples detection rate of 0.73 and an out of range examples detection rate of 0.89. Note that the original model accuracy (without the embedded binary classifier) on out of range examples is 0.0 and 0.298 on adversarial examples.

Important detection rates can be observed as well in the experiment carried out with the Fashion-MNIST dataset (table 6.2). Given the difference among both datasets, we think that similar results would be obtained for other datasets, showing the general applicability of the method.

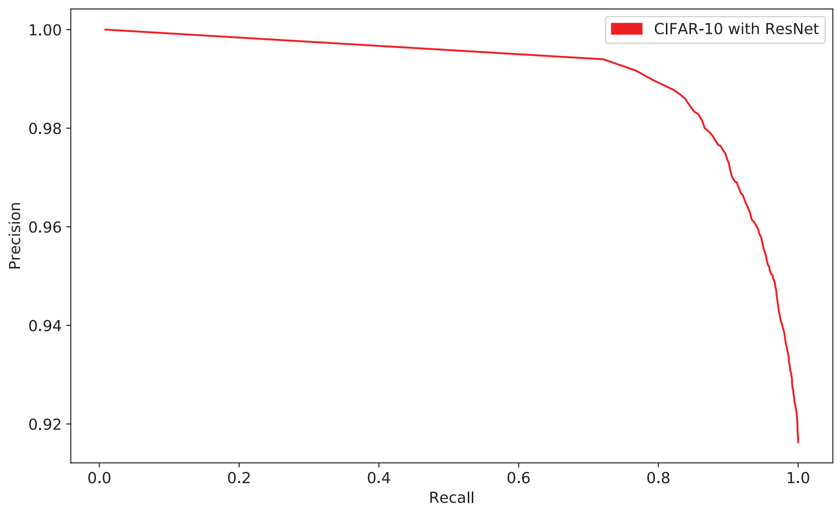


Figure 6.3: Precision recall curve with the entropy threshold as implicit parameter. **CIFAR-10** dataset.

Table 6.1: CIFAR-10 with Resnet

Required precision	Recall	OOR detection rate	Adversarial detection rate	% of manual revision
0.99	0.77	0.98	0.91	29.06%
0.98	0.85	0.89	0.73	19.00%
0.97	0.91	0.82	0.65	14.52%
0.96	0.94	0.68	0.56	10.29%
0.95	0.96	0.59	0.47	7.12%
0.94	0.97	0.53	0.39	5.15%
0.93	0.99	0.37	0.23	2.50%
0.92	0.99	0.14	0.06	0.05%

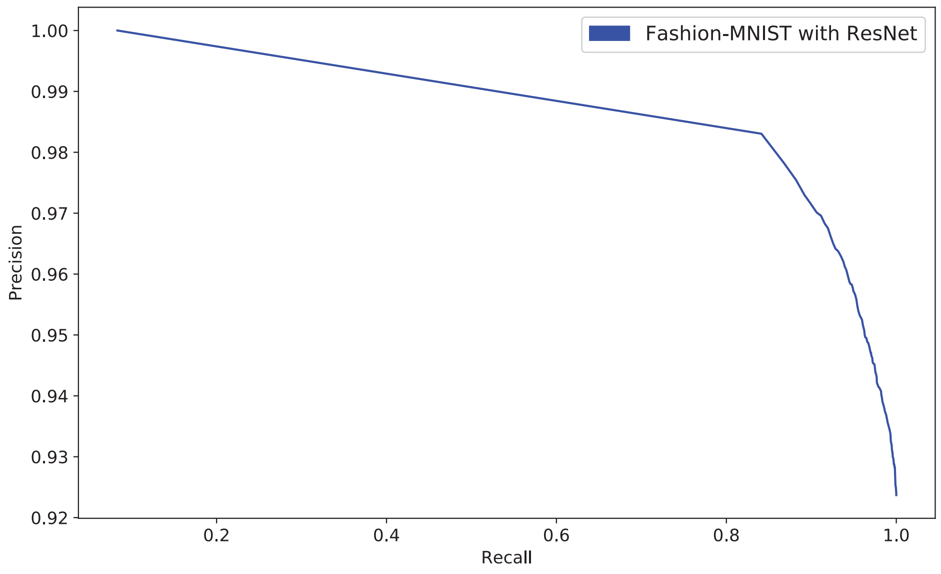


Figure 6.4: Precision recall curve with the entropy threshold as implicit parameter. **Fashion-MNIST** dataset.

Table 6.2: Resnet architecture trained on Fashion-MNIST. Accuracy of original network: 0.926

Required precision	Recall	OOR detection rate	Adversarial detection rate	% of manual revision
0.99	0.08	1.0	1.0	92.31%
0.98	0.84	0.83	0.80	20.95%
0.97	0.90	0.67	0.68	13.70%
0.96	0.94	0.50	0.52	9.49%
0.95	0.96	0.36	0.35	6.42%
0.94	0.98	0.19	0.24	3.49%
0.93	0.99	0.05	0.07	1.11%

6.2 Accuracy improvement with ensembles

The experiments explained in this chapter have been carried out with an stacking architecture as described in section 5.2, trained with datasets CIFAR-10 and fashion-MNIST. The first-stage learners are the ResNet v20 models from the last experiment and the second-stage learner or meta-learner will simply choose, among the first-stage learners, the one with the lower entropy. We analyze the progress of the accuracy vs. the number of learners used for building the ensemble. We find an increase in the accuracy as new components are added, which slows down progressively up to a saturation size of about 10 components. (See figures 6.6, 6.5)

[He *et al.*, 2016] show how it is possible to add additional layers in the ResNet architecture to gain accuracy, avoiding the problem of saturation present in other deep models with an equivalent size. Taking into account the importance of this mentioned paper, we can use its results as a benchmark for our own findings. Accordingly, we compare the gain in accuracy of the previous experiment with the gain in accuracy by adding additional layers to the ResNet architecture from the mentioned paper. The results show that our ensemble strategy obtains a greater accuracy improvement, with a fewer increase in the number of parameters. See table 6.3. With the exception of the last row, that represents our experiments, results on the table are taken from [He *et al.*, 2016].

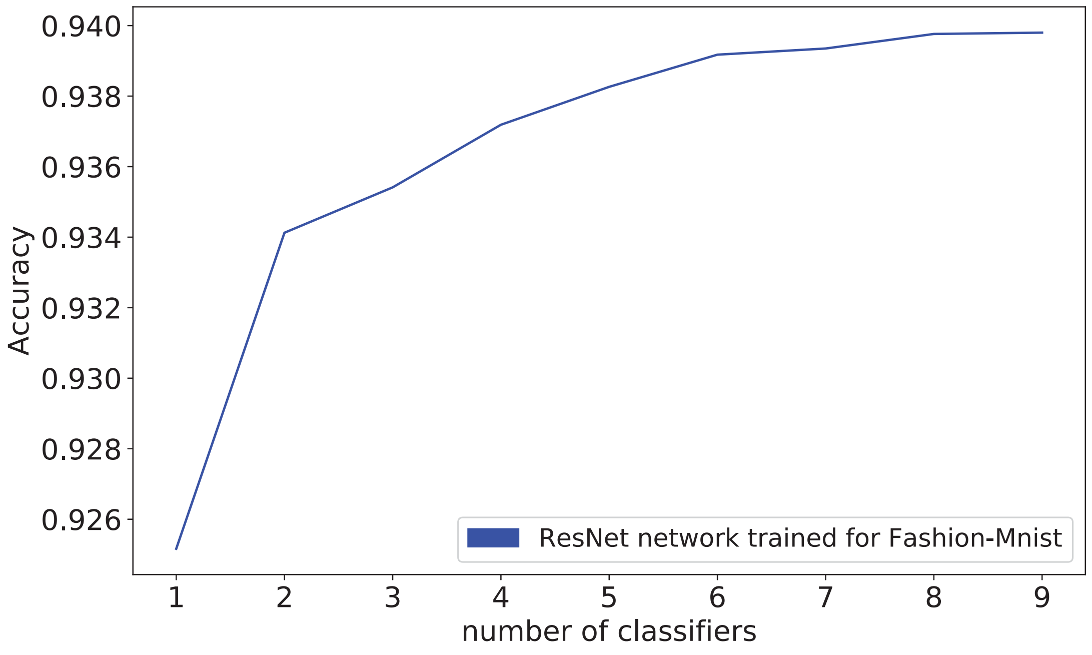


Figure 6.5: Achieved accuracy vs. number of classifiers in the ensemble in a ResNet convolutional network trained on **Fashion-MNIST** dataset.

Table 6.3: accuracy for different Resnet architectures compared to an ensemble. CIFAR10 dataset.

Architecture name	# of layers	#of parameters (thousands)	Accuracy
Resnet 20 v1	20	0.27	91.25%
Resnet 32 v1	32	0.46	92.49%
Resnet 44 v1	44	0.66	92.83%
Resnet 56 v1	56	0.85	93.03%
Resnet 110 v1	110	1.7	93.57%
Resnet 1202 v1	1202	19.4	92.07%
Resnet 20 v1 ensemble (4x)	20(x4)	1.08	**93.70%**

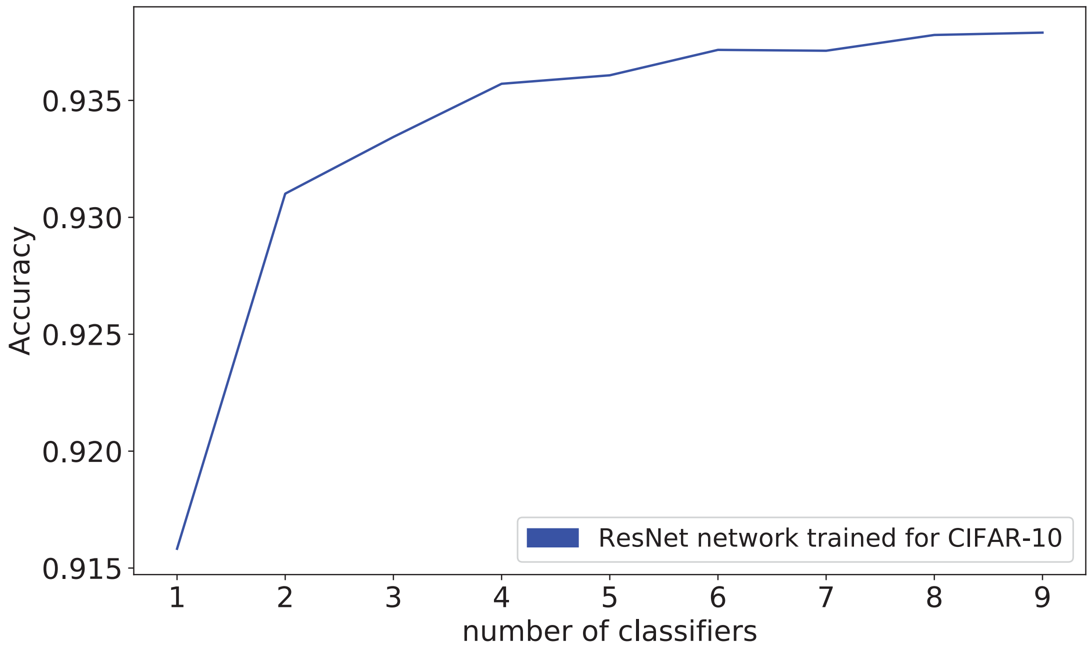


Figure 6.6: Achieved accuracy vs. number of classifiers in the ensemble in a ResNet convolutional network trained on **CIFAR-10** dataset.

Chapter 7

Conclusions and future research

7.1 Conclusions

We have developed a method based on the entropy for detecting prediction errors of deep learning classifiers. Based on it, a system has been built that allows specifying the desired precision, with the trade off of losing automation capacity. For a given precision, the system response under the different scenarios of in range, out of range and adversarial examples, is known beforehand, together with the percentage of predictions that the system will reject and will need manual revision. Based on these facts, the built system can be entitled as more trustworthy than the original classifier.

The entropy method has been also applied in an ensemble of ResNet deep learning classifiers, improving the accuracy of the individual learners. The ensemble is designed with a basic stacking technique where the first stage learners are the mentioned classifiers and the second stage learner, or meta-learner, just makes the final decisions by choosing the learner with the best entropy score. It has been proven that this strategy is more efficient than the method implemented in [He *et al.*, 2016] consisting on adding more layers to a single network. A higher accuracy with less parameters is obtained with the ensemble architecture.

7.2 Future research

The system specifications can be enhanced by performing additional tests under new scenarios, using real images as out of range examples and adversarial examples generated with other existing techniques. This should result in a more foreseeable system and thus more trustworthy. Also, a complementary evaluation of the methods presented in this document could be carried out considering the baseline proposed by [Hendrycks and Gimpel, 2016] to compare out of range example predictors.

As for the ensemble, the second stage learner of the stacking architecture is effective, though basic. Given the multiple opened research lines in the field on ensemble architectures, additional accuracy gains might be obtained applying the entropy method together with more sophisticated ensemble architectures and/or final decision algorithms. Furthermore, a more sophisticated system of detection can be built using a network ensemble in stead of using a single network. Given the improvement achieved in the accuracy using the ensemble, good results can be also expected for the detection of out of range and adversarial examples.

Bibliography

[Aggarwal *et al.*, 2001] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory* (2001), Springer, pp. 420–434.

[Amodei *et al.*, 2016] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565* (2016).

[Amodei *et al.*, 2016-bis] Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning* (2016), pp. 173–182.

[Athalye *et al.*, 2018] Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420* (2018).

[Bachman *et al.*, 2014] Bachman, P., Alsharif, O., and Precup, D. Learning with pseudo-ensembles. In *Advances in Neural Information Processing Systems* (2014), pp. 3365–3373.

[Beyer *et al.*, 2014] Beyer, K., Goldstein, J., Ramakrishnan, R., and Shaft, U. When is "nearest neighbor" meaningful? In *International conference on database theory* (1999), Springer, pp. 217–235.

[Bishop, 2006] Bishop, C. M. *Pattern recognition and machine learning.* springer, 2006.

[Breiman, 1996] Breiman, L. Bagging predictors. *Machine learning 24*, 2 (1996), 123–140.

[Breiman, 2001] Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[Chen & Guestrin, 2016] Chen, T., and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016), ACM, pp. 785–794.

[Collobert & Weston, 2008] Collobert, R., and Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning* (2008), ACM, pp. 160–167.

[Dodds, 2018] Dodds, L. Chinese businesswoman accused of jaywalking after ai camera spots her face on an advert. *The Telegraph* (2018).

[European Comission, 2019] Commission, E. Communication: Building trust in human centric artificial intelligence. Tech. rep., 2019.

[Eykholt *et al.*, 2018] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 1625–1634.

[Gal, 2016] Gal, Y. *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016.

[Gal and Ghahramani, 2017] Gal, Y., and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning* (2016), pp. 1050–1059.

[Goodfellow *et al.*, 2014] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[Goodfellow *et al.*, 2016] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.

[Grush, 2015] Grush, L. Google engineer apologizes after photos app tags two black people as gorillas. *The Verge* (2015).

[Guo *et al.*, 2017] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 1321–1330.

[Hafner *et al.*, 2018] Hafner, D., Tran, D., Irpan, A., Lillicrap, T., and Davidson, J. Reliable uncertainty estimates in deep neural networks using noise contrastive priors. *arXiv preprint arXiv:1807.09289* (2018).

[Haykin, 2009] Haykin, S. S., et al. *Neural networks and learning machines/Simon Haykin*. New York: Prentice Hall,, 2009.

[He *et al.*, 2016] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[Hendrycks and Gimpel, 2016] Hendrycks, D., and Gimpel, K. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136* (2016).

[Hron *et al.*, 2016] Hron, J., Matthews, A. G. d. G., and Ghahramani, Z. Variational bayesian dropout: pitfalls and fixes. *arXiv preprint arXiv:1807.01969* (2018).

[Kendall, 2017] Kendall, A., and Gal, Y. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems* (2017), pp. 5574–5584.

[Krizhevsky *et al.*, 2009] Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Tech. rep., Citeseer, 2009.

[Krizhevsky *et al.*, 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[Lakshminarayanan *et al.*, 2017] Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems* (2017), pp. 6402–6413.

[Lee *et al.*, 2017] Lee, K., Lee, H., Lee, K., and Shin, J. Training confidence-calibrated classifiers for detecting out-of-distribution samples. *arXiv preprint arXiv:1711.09325* (2017).

[Lipton, 2016] Lipton, Z. C. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490* (2016).

[Madry *et al.*, 2017] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[Nicolae, 2018] Nicolae, M.-I., Sinn, M., Tran, M. N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I., and Edwards, B. Adversarial robustness toolbox v0.9.0. *CoRR 1807.01069* (2018).

[Osband *et al.*, 2016] Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems* (2016), pp. 4026–4034.

[Ramsundar *et al.*, 2016] Ramsundar, B., Kearnes, S., Riley, P., Webster, D., Konerding, D., and Pande, V. Massively multitask networks for drug discovery. *arXiv preprint arXiv:1502.02072* (2015).

[Ripley, 2017] Ripley, B. D. *Pattern recognition and neural networks.* Cambridge university press, 2007.

[Shannon 1948] Shannon, C. E. A mathematical theory of communication. *Bell system technical journal 27*, 3 (1948), 379–423.

[Silver *et al.*, 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature 529*, 7587 (2016), 484.

[Srivastava *et al.*, 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[Townsend, 1971] Townsend, J. T. Theoretical analysis of an alphabetic confusion matrix. *Perception & Psychophysics 9*, 1 (1971), 40–50.

[Tumer & Ghosh, 1995] Tumer, K., and Ghosh, J. Theoretical foundations of linear and order statistics combiners for neural pattern classifiers. *IEEE Trans. Neural Networks* (1995).

[Wolpert, 1992] Wolpert, D. H. Stacked generalization. *Neural networks 5*, 2 (1992), 241–259.

[Wolpert, 1996] Wolpert, D. H. The lack of a priori distinctions between learning algorithms. *Neural computation 8*, 7 (1996), 1341–1390.

[Wolpert & Macready, 1997] Wolpert, D. H., Macready, W. G., et al. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation 1*, 1 (1997), 67–82.

[Xiao *et al.*, 2017] Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. *arXiv preprint arXiv:1708.07747* (2017).

[Zhou, 2012] Zhou, Z.-H. *Ensemble methods: foundations and algorithms.* Chapman and Hall/CRC, 2012.