Adversarial Robustness of Model Sets

István Megyeri
University of Szeged
Szeged, Hungary
imegyeri@inf.u-szeged.hu

István Hegedűs
University of Szeged
Szeged, Hungary
ihegedus@inf.u-szeged.hu

Márk Jelasity
University of Szeged and
MTA-SZTE Research Group on AI
Szeged, Hungary
jelasity@inf.u-szeged.hu

***Abstract*—Machine learning models are vulnerable to very small adversarial input perturbations. Here, we study the question of whether the list of predictions made by a list of models can also be changed arbitrarily by a single small perturbation. Clearly, this is a harder problem since one has to simultaneously mislead several models using the same perturbation, where the target classes assigned to the models might differ. This attack has several applications over models designed by different manufacturers for a similar purpose. One might want a single perturbation that acts differently on each model; like only misleading a subset, or making each model predict a different label. Also, one might want a perturbation that misleads each model the same way and thereby create a transferable perturbation. Current approaches are not applicable for this general problem directly. Here, we propose an algorithm that is able to find a perturbation that satisfies several kinds of attack patterns. For example, all the models could have the same target class, or different random target classes, or target classes designed to be maximally contradicting. We evaluated our algorithm using three model sets consisting of publicly available pre-trained ImageNet models of varying capacity and architecture. We demonstrate that, in all the scenarios, our method is able to find visually insignificant perturbations that achieve our target adversarial patterns.** [1]

***Index Terms*—adversarial examples, multi-model attack, model set attack, DeepFool, deep neural networks**

I. Introduction

Several studies have demonstrated that current deep learning models are highly sensitive to small adversarial input perturbations [1], [2]. The original formulation of the problem assumes that we are given a model and a correctly classified example. The attacker wishes to find a minimal input perturbation that results in the prediction of any wrong label (untargeted attack) or a given desired label (targeted attack). In the past few years, a large number of methods have been proposed to create better adversarial examples [3], [4] and to provide defense mechanisms [5], [6]. Yuan et al. provide a recent overview [7].

We study a more general version of this problem, where we are given a set of models trained on the same multi-class classification problem, as well as an input example. We assume that our set contains independently trained standalone models. That is, our focus is *not* on model ensembles. We could envisage the models to be products of different manufacturers.

This study was supported by the National Research, Development and Innovation Office of Hungary through the Artificial Intelligence National Excellence Program (grant 2018-1.2.1-NKP-2018-00008) and by grant TUDFO/47138-1/2019-ITM of the Ministry for Innovation and Technology, Hungary.

[1]To encourage reproducible research, the code of our algorithm is made available at http://github.com/istvanmegyeri/ARMS

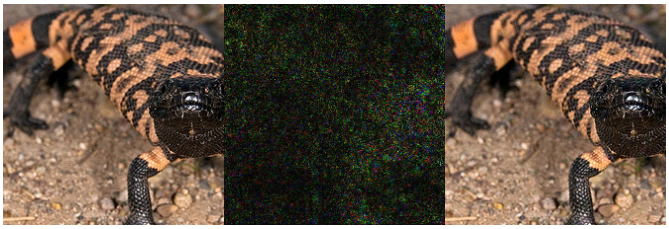

Fig. 1. Left: original image (prediction: 'Gila monster'); Middle: multi-model perturbation; Right: adversarial image (predictions: MobileNetV2: 'bison', MobileNet: 'balloon', NASNetMobile: 'pole', DenseNet121: 'acorn', DenseNet169: 'washbasin, handbasin, washbowl, lavabo, wash-hand basin', DenseNet201: 'custard apple').

Nevertheless, our results are applicable to any set of models. We assume that the models are completely known; in other words, we are concerned with the white box scenario.

Our formulation allows for a wide variety of attacks on a given model set. The attacker may want every model to make the same mistake or (if, for example, hired by one of the manufacturers) to make different mistakes. The scenario where all of the models are made to predict the same wrong label is also related to the problem of finding adversarial examples for ensembles of models in the hope that these examples will also fool other unseen black box models [6], [8], [9]. In our version, however, every single model will make the given wrong prediction; not only the ensemble as a whole, where it might be enough to mislead, for example, the majority of the models (depending on the ensemble decision method).

An attacker may also be a nihilist who wants to achieve chaotic behavior. One way of doing this would be to make every model predict different random labels for the same input. One might think that a single perturbation that satisfies such harsh conditions should be extremely hard to find or might not even exist. Figure 1 shows such a perturbation, found by our algorithm. In a recent work, Song et al. investigated random target labels [10] in a multi-label classification setting. However, our setting is more challenging because we have a set of models trained on the same multi-class classification task. For this reason, the intersection of arbitrary classes is expected to be smaller than in a multi-label problem.

We propose a heuristic iterative algorithm to solve our multi-model adversarial problem. The algorithm is inspired by the DeepFool method [3] in that we also guide our search with the help of linear approximations of decision boundaries. We evaluated our method over the ImageNet dataset [11] using three sets of pre-trained models. We selected model sets so

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

that we could evaluate different aspects of model diversity such as architecture and capacity. Model capacity is interesting to consider, since it has been shown that capacity alone could increase the robustness of individual models [5]. The diversity of the model architectures within the set is another important factor. For example, in [9], Pang et al. used the ensemble of diverse models to train a robust ensemble classifier. We expect that the diversity and the capacity of the models play a key role in robustness, and a set that is diverse along both dimensions is harder to attack.

We demonstrate that our algorithm can find feasible adversarial perturbations that fool all the models according to the given pattern in all the scenarios we examined. The adversarial target patterns include predicting the same wrong label, predicting different randomly selected labels, and predicting a set of labels that are designed to be maximally inconsistent. We found that model sets that include models with different architectures have a somewhat higher robustness than sets with similar architectures but with different capacity. However, in each case, the perturbations we found are imperceptible to the human eye.

In a recent paper [12], we made preliminary steps towards tackling the multi-model problem. Our original contributions here are the following:

1) Formalizing the targeted multi-model adversarial perturbation problem.
2) Proposing and evaluating several novel algorithm variants for solving the multi-model adversarial perturbation problem. These include the application of the pseudo-inverse of the constraint matrix, the application of the aggregated gradient instead of the maximal distance direction, the introduction of a step size limit to reduce the perturbation size, and the introduction of a candidate class list to manage problems with many classes.
3) Evaluating the algorithm over ImageNet using pre-trained publicly available deep networks.
4) Evaluating several adversarial patterns including the random label assignment and two variants of 'hard' label assignments where the idea is to assign different target labels to different models so that the required perturbation is maximally inconsistent.

The outline of the paper is as follows. In Section II we describe our algorithms in detail. In Section III we present several designs for creating adversarial target pattern. In Section IV we present our experimental evaluation. Section V concludes the paper.

II. Attacking Model Sets

Let us first introduce our notations. As mentioned in the Introduction, the problem is defined over a set of models and an example to perturb. The example is given in the form (x, y), where $x \in \mathbb{R}^d$ is the feature vector, and $y \in \mathcal{C}$ is its true label. The set $\mathcal{C}$ contains the class labels and $\mathcal{C}_{adv} = \mathcal{C} \setminus \{y\}$ is the set of possible adversarial target labels for x.

The set of multi-class models is given as $\mathcal{F} = \{f_1, \ldots, f_m\}$, where $f_i : \mathbb{R}^d \to \mathbb{R}^{|\mathcal{C}|}$. That is, the models have one output for every possible class label. We assume that all the models are trained over the same multi-class problem. The classification of a given input x by a model f_i is given by $k_i(x) = \arg\max_j f_{i,j}(x)$, where $f_{i,j}(x)$ is the jth element of the vector $f_i(x)$.

Although not strictly required by the framework, we note that in our implementation we used the unnormalized output (that is, the output without the softmax normalization) as our models f_i. When it is possible, this makes sense, because the normalization simply makes the gradients flatter without changing the ranking of the labels.

The predicted labels of the models are given by $K(x) = (k_1(x), \ldots, k_m(x)) \in \mathcal{C}^m$. In this study, we define the set $\mathcal{C}^m_{adv}$ as the set of all the possible adversarial patterns for the model set $\mathcal{F}$. Note that this way, we require here that all the models are assigned an adversarial label. This is not a severe restriction, one could also allow or even require some models to retain their true labels. An attacker will probably have further specific restrictions on the desired target patterns, so the set of desired adversarial patterns are given as a subset of all the patterns as $\mathcal{P} \subseteq \mathcal{C}^m_{adv}$. Formally, we are looking for a perturbation r^* that has minimal L_2 norm and makes the classifiers predict one of the desired patterns:

$$r^* = \arg\min_r \|r\|_2, \text{ subject to } K(x + r) \in \mathcal{P}. \tag{1}$$

The constraint of this optimization problem in (1) can be unrolled as a system of equations for a given pattern $p \in \mathcal{P}$ as

$$\begin{aligned} k_1(x + r) &= p_1 \\ &\vdots \\ k_m(x + r) &= p_m. \end{aligned} \tag{2}$$

We can transform this system of equations into a system of inequalities using the fact that the equations require that the adversarial label correspond to the maximal element in the model's output. Accordingly, for the ith equation $k_i(x+r) = p_i$, let $\hat{k}_i(x) = \arg\max_{j \neq p_i} f_{i,j}(x)$ be the label that is the most likely label among the labels other than p_i. This gives us the equivalent system of inequalities

$$\begin{aligned} f_{1,\hat{k}_1(x+r)}(x + r) &\leq f_{1,p_1}(x + r) \\ &\vdots \\ f_{m,\hat{k}_m(x+r)}(x + r) &\leq f_{m,p_m}(x + r), \end{aligned} \tag{3}$$

where the equality holds when a point $x + r$ is exactly on the decision boundary between classes p_i and $\hat{k}_i(x + r)$. This can be considered as a generalization of the formalization of the DeepFool [3] algorithm where, instead of a system of inequalities, we have just a single inequality. In other words, in the model set there is only one model.

Due to the nonlinearities of the functions $f_{i,j}(x + r)$, this problem cannot be solved directly. As in the DeepFool algorithm, we substitute these functions with their first order approximations around x, that is, $f_{i,j}(x + r) \approx f_{i,j}(x) +$

Algorithm 1 Multi-model adversarial perturbation

Input: example x, models $\mathcal{F}$, adversarial patterns $\mathcal{P}$
$x_0 \leftarrow x$
$i \leftarrow 0$
while $i < i_{max}$ and $K(x_i) \notin \mathcal{P}$ **do**
  for $p_k \in \mathcal{P}$ **do**
    $r_k \leftarrow$ approximateQP(x_i, p_k)
  end for
  $r \leftarrow r_{\arg\min_k \|r_k\|_2}$ ▷ r_k with the smallest norm
  $r \leftarrow \min(\eta/\|r\|_2, 1) \cdot r$ ▷ enforce $\|r\|_2 \leq \eta$
  $x_{i+1} \leftarrow x_i + r$
  $i \leftarrow i + 1$
end while
return x_i ▷ the perturbed input

$\nabla f_{i,j}(x)^T \cdot r$, which results in the following system of inequalities:

$$\begin{aligned} \nabla(f_{1,\hat{k}_1(x)}(x) - f_{1,p_1}(x))^T \cdot r &\leq f_{1,p_1}(x) - f_{1,\hat{k}_1(x)}(x) \\ &\vdots \\ \nabla(f_{m,\hat{k}_m(x)}(x) - f_{m,p_m}(x))^T \cdot r &\leq f_{m,p_m}(x) - f_{m,\hat{k}_m(x)}(x), \end{aligned} \tag{4}$$

which is a set of linear constraints. The objective of the optimization problem is quadratic, which means that we have a quadratic programming (QP) problem.

A naive view would be to solve this problem with an efficient QP solver, and—since the linear constraints only approximate the actual constraints—iterate this procedure until the original set of constraints is satisfied. Using a QP solver, however, introduces certain problems. First, although efficient solvers are known, we might need to run them many times, which might become inefficient. Second, if there is no feasible solution for the linear approximation, the original problem might still have one, so we should create new approximations for each iteration. For these reasons, instead we used three heuristics to approximate the solution of the QP problem. We discuss these later on in more detail.

We are now ready to present the pseudocode of the algorithm, represented as Algorithm 1. We assume that we are given an example x. The models in $\mathcal{F}$ have to predict one of the patterns in $\mathcal{P}$. Recall that if there are more than one pattern in $\mathcal{P}$, satisfying any of these patterns is considered a success. The outer loop runs until this goal is met (or we reach the maximal number of iterations).

The inner loop iterates through the patterns, and calculates the heuristic solution of the QP that is defined by the particular pattern. After calculating these approximate solutions r_k, the one with the smallest norm is selected. If the magnitude of this perturbation is larger than the allowed maximum step size η, then the perturbation is normalized. This step offers some protection to the algorithm in cases where the linear approximation has a large error. This perturbation size restriction technique is partly related to trust region methods. Here, we use a fixed η parameter that controls the size of the trusted region in a static manner. Changing η dynamically could offer further improvements [13].

A. Heuristics to Solve the Inner QP Problem

Let us now describe three implementations of the method APPROXIMATEQP that is used in the inner loop of Algorithm 1. Let us first normalize the inequalities in (4) by dividing both sides by $\|\nabla(f_{i,\hat{k}_i(x)}(x) - f_{i,p_i}(x))\|_2$. Let us introduce a new notation to represent the normalized inequalities. On the left hand side, let

$$w_i = \frac{\nabla(f_{i,\hat{k}_i(x)}(x) - f_{i,p_i}(x))}{\|\nabla(f_{i,\hat{k}_i(x)}(x) - f_{i,p_i}(x))\|_2} \tag{5}$$

and on the right hand side, let

$$f'_{i,p_i} = \frac{f_{i,p_i}(x) - f_{i,\hat{k}_i(x)}(x)}{\|\nabla(f_{i,\hat{k}_i(x)}(x) - f_{i,p_i}(x))\|_2}. \tag{6}$$

With these notations, equation i becomes $w_i^T \cdot r \leq f'_{i,p_i}$.

Note that the value f'_{i,p_i} represents the gap between the class that is currently predicted by model f_i and the desired adversarial target class p_i. Thus, we know that at the beginning of the attack $f'_{i,p_i} \leq 0$ and the equality holds exactly when the predicted class is already the target class. From now on, all the heuristics below will ignore the models where the target class is already predicted, and the algorithms will work only with those equations that still have a negative right hand side.

1) MAX: Here, the idea is that we first identify the model that has the largest gap between its target class and the current class. We then approximate the minimal perturbation that closes this gap, that is, that 'fixes' this model. The idea is that this way we first solve the hardest model and then gradually adjust for the rest of the constraints. The approximation takes the linear approximation of the model, for which the minimal perturbation can be computed in closed form. The formulation of this heuristic is

$$\begin{aligned} j &= \arg\max_{1 \leq z \leq m} -f'_{z,p_z} \\ r_{max} &= -f'_{j,p_j} w_j. \end{aligned} \tag{7}$$

Note that this computation is very similar to that of DeepFool, since here we have a single model to consider (the one with the maximal gap), and on that model we essentially run a DeepFool step.

2) AVG: Here, we compute a direction that takes into account all the models, instead of just choosing the one with the maximal gap. Recall, that the vector w_i represents the direction of the minimal perturbation required to 'fix' model i, and $\|w_i\|_2 = 1$ due to the normalization. We first compute the average of these directions, weighted by the gap values. We then return a perturbation vector that points in this average direction. The length of the returned vector is computed to be the maximum of the gap values. The reason is that this way we are guaranteed not to overshoot, because this step size is the minimal step size that potentially changes all the predictions. If we use a smaller step size, there will be at least one model that will not predict the target label. The formulation is

$$\begin{aligned} w_{avg} &= \textstyle\sum_{z=1}^{m} -f'_{z,p_z} w_z \\ j &= \arg\max_{1 \leq z \leq m} -f'_{z,p_z} \\ r_{avg} &= -f'_{j,p_j} \frac{w_{avg}}{\|w_{avg}\|_2}. \end{aligned} \tag{8}$$

3) PINV: When we have only one model, the minimal perturbation is computed as one step of the Newton method, as done by DeepFool and by the MAX heuristic above as well. We could also try to generalize this idea to multiple models by taking the set of inequalities in (4) and demanding that equality holds in each inequality. Since the number of variables will be usually larger than the number of equalities, the system will be underdetermined. We can then still solve this set of equalities using the pseudo-inverse technique as applied in [10].

This heuristic will return a perturbation that points towards the intersection of all the decision boundaries, which may or may not be the smallest perturbation among those that satisfy (4). In pathological cases, this intersection might be extremely far from the optimal perturbation direction. In fact, there might not even be such an intersection. Still, we include this algorithm for comparison with [10]. Also this algorithm could potentially be fast (ignoring pathological corner cases), since in every iteration we try to solve every constraint.

Consider the matrix $W = (w_1, \cdots, w_m)^T$ and vector $F = (-f'_{1,p_1}, \cdots, -f'_{m,p_m})^T$. The system of equations is then $W \cdot r = F$, so the perturbation returned by the PINV algorithm is given by

$$r_{pinv} = W^+ \cdot F, \tag{9}$$

where $W^+ = (W^T W)^{-1} W^T$ (assuming W has full rank) is the pseudo-inverse of W. Note that it is known that r_{pinv} will be the minimum norm solution if there is a solution, and it will represent an approximation with minimal error if there is no solution [14].

B. Time complexity

The number of iterations of the algorithm depends on many factors and hyperparameters. We evaluate this experimentally later on. Here, we focus on the time complexity of a single iteration. To perform one iteration, the linear approximation in (4) needs to be calculated for every $p \in \mathcal{P}$. For each pattern p, we need to compute the prediction and the gradient of each model, which means that we need to propagate one forward pass and one backward pass through every network, if the models are feedforward neural networks. Thus, the cost of one iteration is $2 \cdot m \cdot |\mathcal{P}|$ network propagations. If $\mathcal{P}$ has only one element then the time complexity depends only on m (the number of models). Note that these calculations can be parallelized across the models as well as the patterns, and only the forward pass needs to be performed before the backward pass.

III. Adversarial Target Patterns

Our algorithm was formulated as a targeted attack, where the acceptable adversarial patterns are given in the pattern set $\mathcal{P}$. This set might contain a single pattern, which is similar to the classical targeted attack. Although it would be possible to formulate a modified algorithm for the untargeted case, one can achieve a similar effect by including several carefully selected patterns in the target set.

In the following, we describe and motivate our four different designs for the pattern set that we will use in our experimental evaluation. We assume that we are given a fixed input x and the patterns are designed as a function of this input. Our patterns include three targeted patterns and an untargeted one.

Some of the designs are based on a ranking of class labels $\mathcal{C}$, based on the output of all the models for a given input x. For a given input, each individual model defines a ranking of the class labels based on the ordering of the values of the corresponding output elements. The highest ranking class is predicted by the model. The idea is that we wish to identify those class labels that are 'similar' to the true class label and hence that are ranked high by most of the models. Further, we are also interested in those labels that are treated as very irrelevant by most of the models.

The ranking of the labels can be defined as a suitable aggregation of the rankings of the individual models. Formally, the ranking of a class label $c \in \mathcal{C}$ for a given example x and a classifier f_i is denoted by $q_{i,x}(c) \in \{0, \ldots, |\mathcal{C}| - 1\}$. In other words, $q_{i,x}(c)$ is the rank of label c in the ranking defined by $f_i(x)$. For the predicted class label we have $q_{i,x}(k_i(x)) = 0$. Note that any ranking aggregation could be used to determine the common ranking. Here we used the Rank Product [15] aggregation method, where the ranking of a label c is defined by the ordering of the geometric mean rankings

$$RP_x(c) = \prod_{j=1}^{m} q_{j,x}(c)^{1/m}, \quad \forall c \in \mathcal{C} \tag{10}$$

A. Random

$\mathcal{P}_{random}$ contains only one element, that is, it defines a targeted attack. This single pattern contains randomly generated adversarial class labels for all of the models. This attack should be hard because the perturbed input should end up in the intersection of the different unrelated classes of the different models.

B. Reverse

Our goal here is to define a target pattern that is even harder than the random pattern. $\mathcal{P}_{reverse}$ will also contain only a single pattern. To create this pattern, we used the ranking of the class labels described above. From this ranking, we selected the label that ranks the lowest, namely the label that is the most irrelevant among the model set. We then demand that the pattern contain this label for all the models.

C. Diverse

As another attempt to define a pattern that is maximally hard, here $\mathcal{P}_{diverse}$ has one pattern that contains labels that rank low and, in addition, that are also inconsistent with each other. We again use the low end of the ranking of the labels but this time we consider the last 10 labels. Next, we compute the adversarial direction for this ten labels for every model, using a single linear approximation (DeepFool) step. We now have 10 directions for all the models. Out of this set, for every model we select the direction such that the set of selected directions over all the models form a maximally diverse set. We measured the diversity of a given candidate pattern with the help of the average cosine similarity of every pair of directions. The pattern with the lowest value is selected. In our evaluation, we performed an exhaustive search to find the most diverse pattern.

TABLE I
NETWORKS USED IN THE EVALUATION

Model	Parameters	Depth	Correctly Classified
MobileNetV2 [16]	3.5M	88	7153
MobileNet [17]	4.2M	88	7257
NASNetMobile [18]	5.3M	-	7832
DenseNet121 [19]	8.1M	121	7729
DenseNet169 [19]	14.3M	169	8095
DenseNet201 [19]	20.2M	201	8331

TABLE II
MODEL SETS

Model set	Correctly Classified
Mobile	53
Dense	74
All	52

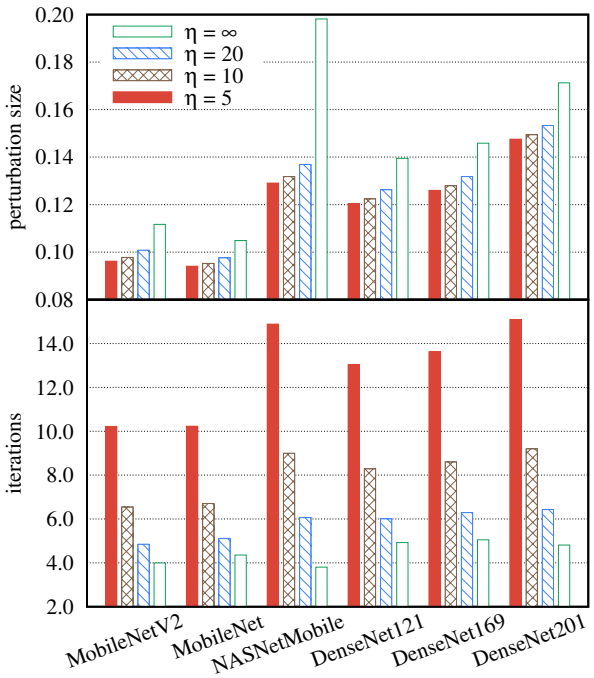

Fig. 2. Perturbation size and number of iterations for the individual models as a function of maximum step size (η). The models are shown in the order of increasing capacity.

D. Consistent

Here, we define an untargeted attack. This case is exceptional for two reasons. First, we include more than one pattern in $\mathcal{P}_{consistent}$. Second, and most importantly, in this case the pattern set will be dynamic; that is, we will update the set of target patterns in every iteration using the method described here. For clarity of presentation, this step is not included in the pseudocode of Algorithm 1.

The patterns can just target the same adversarial label for every model. In other words, all the models are required to predict the same, adversarial label. We used the top 10 adversarial labels in each cycle. The ranking of the labels is updated in each iteration using the Rank Product method and the (already perturbed) input x_i.

IV. EXPERIMENTS

We did experiments on models trained over the ImageNet [11] dataset that contains RGB images with varying sizes. The images are labeled with one of 1000 class labels. In the preprocessing stage, a 224×224 image was cropped from the middle of every original image.

Table I lists the pre-trained models we used along with references. The table contains basic information about the architecture of the networks. To evaluate our multi-model algorithm, we created model sets using these individual models. The mobile set includes the three mobile networks. The members of the mobile set have a similar capacity, that is, they have a similar number of parameters. However, they have rather different network architectures. MobileNet uses depthwise separable convolutions and MobileNetV2 uses residual connections as well. NasNetMobile contains non-human-designed blocks. The dense set contains the three variants of DenseNet. These models have a similar architecture but have a different capacity. Lastly, we also experimented with the union of the mobile and dense sets that contain all the six models. We will refer to this set as 'All'.

During our experiments, we evaluated the individual models as well as the three model sets. In our evaluation, we worked with a set of 10,000 randomly selected images taken from the training set for the individual models. The subset was the same for every model. Out of this set of examples, we removed those examples that were misclassified by the respective model. Table I shows the number of examples that were classified correctly. These examples form the evaluation set for the given model. Due to their larger cost, the model sets were evaluated on a smaller set, that contains 100 randomly selected examples taken from the training set. Here, we also used the same examples for every model set. Table II shows the number of examples that were classified correctly by all the members of the given set. These examples form the evaluation set for the given model set.

Our main figure of merit is *perturbation size*. We define perturbation size as the L_2 norm of the adversarial perturbation found, normalized by $\sqrt{d}$, where $d = 224 \times 224 \times 3$ is the input dimension. We normalize by $\sqrt{d}$ because in the case of image data, this way we characterize the average perturbation of each pixel irrespective of the resolution of the image, which is a more natural measure. Recall, that each input feature has a value in the range $[0, 255]$.

The properties of the individual models can be seen in Figure 2. Note that for individual models, our algorithm is equivalent to DeepFool when $\eta = \infty$. All the attacks on all the correctly classified examples were successful, irrespective of the value of η. Clearly, the perturbation size increases with model capacity. When setting a smaller η, we get a smaller perturbation, but at the cost of more iterations. Interestingly, the effect of η is much stronger on NASNetMobile than on any other model. In this case, we can get a marked improvement over the DeepFool algorithm. This is interesting,

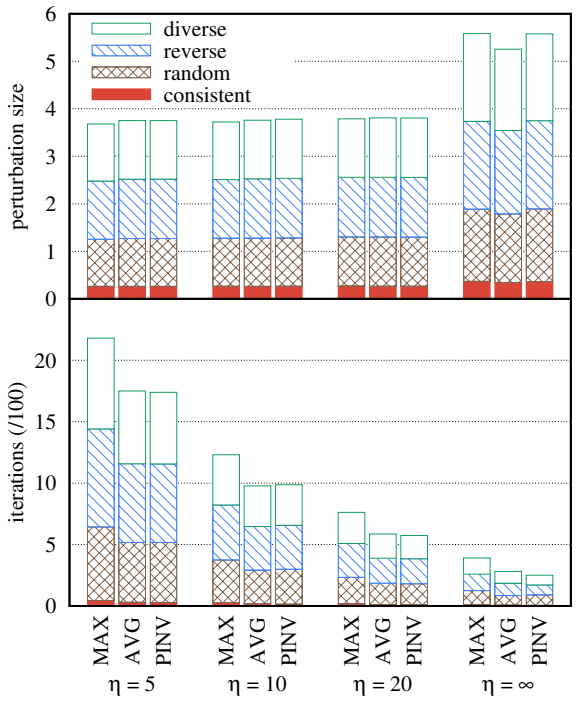

Fig. 3. Perturbation size and number of iterations for the mobile set as a function of the maximum step size (η) and QP solver heuristic.

because DeepFool is often used to compute an upper bound on the sensitivity of models, and in this case this upper bound could be improved significantly. In general, setting a smaller η improves the perturbation size in each case to some extent.

Next, we evaluated our multi-model algorithms on the mobile set to test the effect of η, and the different QP solver heuristics. The results are shown in Figure 3. We specified $i_{max} = 10,000$. Like before, all the attacks were successful, although the required number of iterations is significantly larger than in the case of the individual models. A lower η results in a smaller perturbation and an increased iteration number.

The PINV and the AVG heuristics need fewer iterations, as we expected. This is because these heuristics take into account all the models in the model set in each iteration. We can see that PINV results in the largest perturbation, again, as expected, although the difference from the alternative heuristics is surprisingly small.

For the other two model sets (dense and all), we performed our experiments with a fixed $\eta = 10$ maximum step size because, based on Figure 3, it offers a good compromise between perturbation size and cost. The perturbation size and the number of iterations over the three model sets and the four attack patterns can be seen in Figure 4. The three heuristics are shown separately. All the attempted attacks were successful. The perturbation size of every model set for the four patterns is larger than that for the individual models. Also, significantly more iterations are required.

The consistent pattern leads to the smallest perturbation size and the random and reverse patterns give a larger perturbation,

TABLE III
UNTARGETED INDIVIDUAL MODEL ATTACK WITH $\eta = 10$

Model	Mean iterations
MobileNetV2	6.5
MobileNet	6.7
NASNetMobile	9
DenseNet121	8.3
DenseNet169	8.6
DenseNet201	9.2

TABLE IV
UNTARGETED MODEL SET ATTACK WITH $\eta = 10$

Model set	MAX mean iter	AVG mean iter	PINV mean iter
Mobile set	25.9	16.9	14.9
Dense set	25.9	16.6	14.6
All	57.9	26.9	24.6

as expected. Surprisingly, the diverse pattern is not consistently harder than the reverse pattern. In terms of the number of iterations, there are significant differences among the attack patterns. The attacks that are designed to be hard always require orders of magnitude more iterations than the easiest (untargeted) attack.

It is interesting that the mobile set requires a larger perturbation than the dense set. We expected the opposite, because the attack seemed to be harder if the models in the set are similar, due to the requirement of unrelated classes having a non-empty intersection. Combining all the six models further increases the perturbation size, but it is still less than 1% on average for an input feature. As we will see later, such a perturbation is still imperceptible.

To better illustrate the difference between the number of iterations required to compute the attack for individual models and model sets, we present the average iteration number for these two cases in tables III and IV. We illustrate the untargeted case, that is, the attack pattern was $\mathcal{P}_{consistent}$ for the model set, and a simple untargeted DeepFool attack was used for the individual attacks. From these tables, we see that the required number of iterations appears to be proportional to the number of models in the set.

To illustrate the perturbations that our algorithm creates, we include an example for all the combinations of our three model sets and four attack patterns in figures 5, 6, 7 and 8. The examples we include are the ones that have the largest perturbation among the example inputs. The applied heuristic was MAX, and we set $\eta = 10$. In all the images, the top row contains the original images, the middle row contains the perturbation and the perturbed images are in the bottom row. In the patterns, the order of the models is the same as in Table I.

V. CONCLUSIONS

Here, we introduced an iterative algorithm to find small adversarial perturbations that fool multiple models simultaneously in a given pattern. This problem formulation has many interesting applications, such as the generation of transferable adversarial examples as well as generating a single perturbation such that all the models in a given model set predict

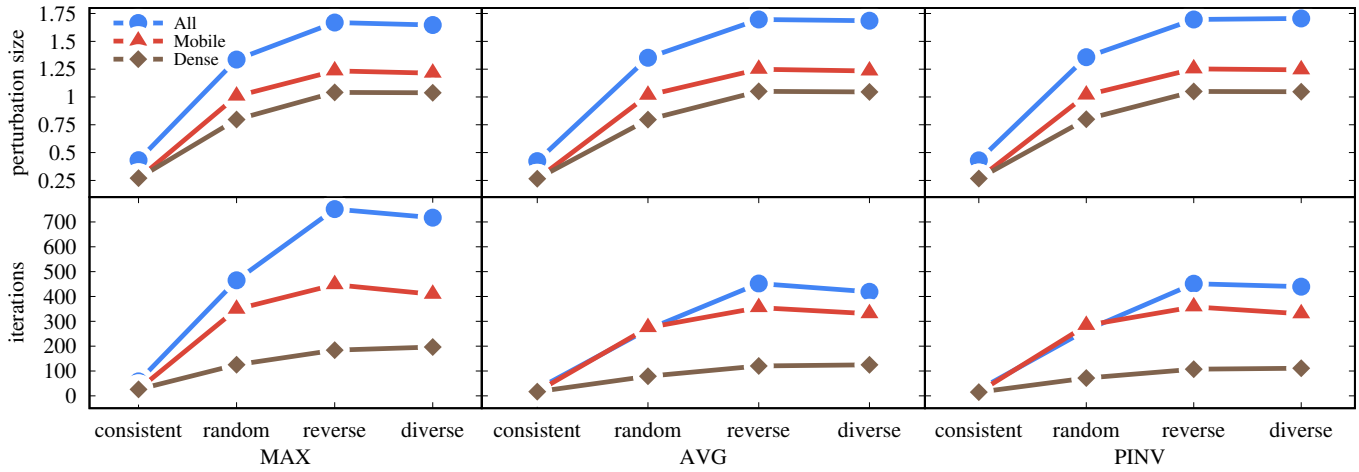

Fig. 4. Perturbation size and iterations for all the four attack patterns and the three QP solver heuristics ($\eta = 10$).

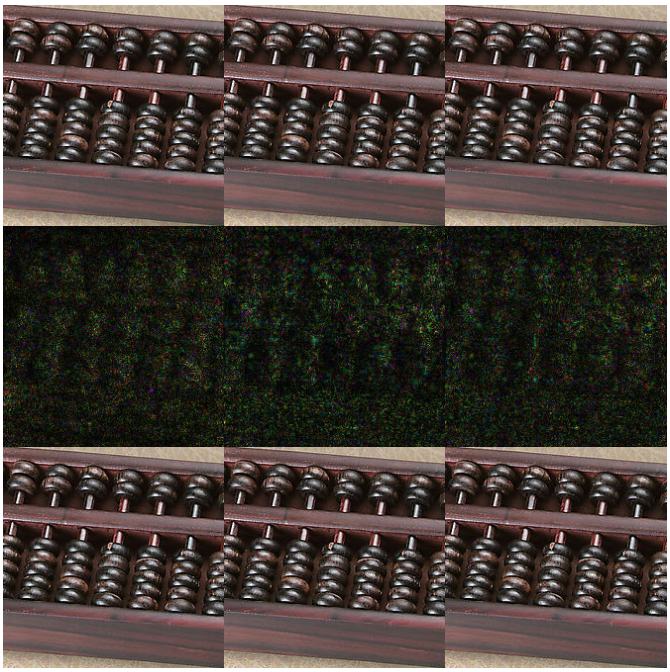

Fig. 5. The consistent attack pattern over the mobile set (abacus → dumbbell), dense set (abacus → corn) and all the models (abacus → dumbbell).

Fig. 6. The random attack pattern over the mobile set (crib → [llama, thunder_snake, Norwich_terrier]), dense set (Australian_terrier → [cornet, lycaenid, malinois]), and all the models (abacus → [centipede, Pembroke, Band_Aid, bow-tie, EntleBucher, coyote, poncho]).

specified, different classes. The latter scenario allows us to explore the decision boundaries of the model set from a new perspective.

The algorithm can be regarded as a generalization of the DeepFool method to model sets. Also, we improved the DeepFool algorithm itself by adding the step size parameter. We evaluated our algorithm on three model sets using four attack patterns over the ImageNet database. We found that the algorithm produces small and successful perturbations reliably in all the attack scenarios we examined. Perhaps the most interesting result is that imperceptible adversarial perturbations were found even when the labels were selected to make the problem as hard as possible. This was surprising to us, even in the light of the vast literature on adversarial attacks.

The perturbation sizes over the three model sets offered some interesting insights as well. The set with different model architectures (mobile set) needed somewhat larger perturbations, but we expected just the opposite. Increasing the size of the model set increased perturbation size as well. Nevertheless, all the perturbations we found are imperceptible to the human eye.

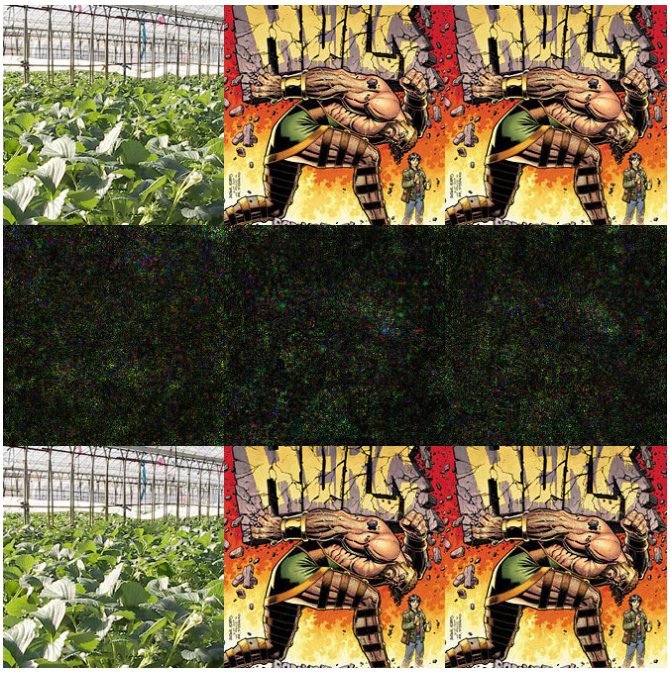

Fig. 7. The reverse attack pattern over the mobile set (greenhouse → projector), dense set (comic_book → albatross) and all the models (comic_book → mongoose).

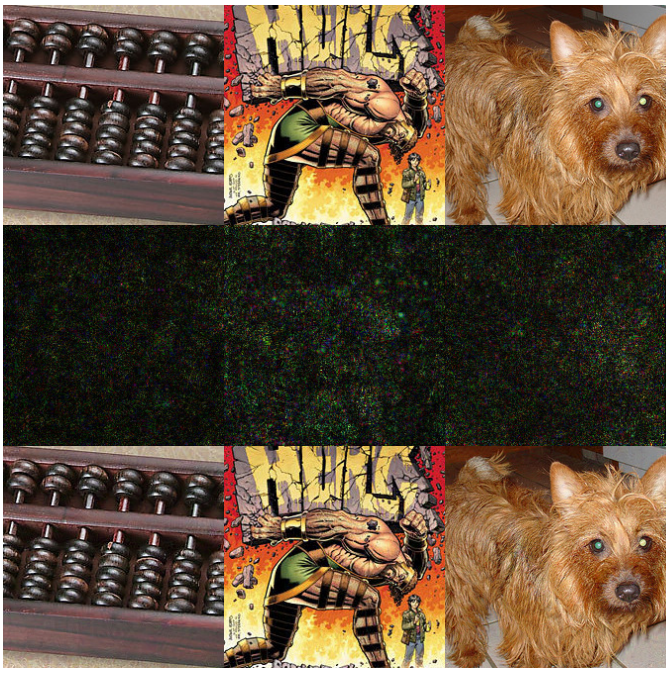

Fig. 8. The diverse attack pattern over the mobile set (abacus → [soft-coated-wheaten_terrier, soft-coated_wheaten_terrier, apron]), dense set (comic_book → [sturgeon, black_stork, capuchin]), and all the models (Australian_terrier → [Saluki, borzoi, black_stork, Saluki, gorilla, kuvasz]).

REFERENCES

[1] I. J. Goodfellow and J. S. C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd Intl. Conf. on Learning Representations (ICLR)*, 2015. [Online]. Available: https://arxiv.org/abs/1412.6572

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd Intl. Conf. on Learning Representations (ICLR)*, 2014. [Online]. Available: http://arxiv.org/abs/1312.6199

[3] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *The IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 2574–2582.

[4] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 39–57. [Online]. Available: https://doi.org/10.1109/SP.2017.49

[5] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *6th Intl. Conf. on Learning Representations (ICLR)*, 2018. [Online]. Available: https://openreview.net/forum?id=rJzIBfZAb

[6] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *Proc. 6th International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: https://openreview.net/forum?id=rkZvSe-RZ

[7] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Trans. Neural Networks and Learning Syst.*, vol. 30, no. 9, pp. 2805–2824, Sep. 2019.

[8] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," in *Proc. 5th International Conference on Learning Representations (ICLR)*, 2017. [Online]. Available: https://openreview.net/forum?id=Sys6GJqxl

[9] T. Pang, K. Xu, C. Du, N. Chen, and J. Zhu, "Improving adversarial robustness via promoting ensemble diversity," in *Proceedings of the 36th International Conference on Machine Learning, (ICML)*, 2019, pp. 4970–4979. [Online]. Available: http://proceedings.mlr.press/v97/pang19a.html

[10] Q. Song, H. Jin, X. Huang, and X. Hu, "Multi-label adversarial perturbations," in *2018 IEEE International Conference on Data Mining (ICDM)*, Nov 2018, pp. 1242–1247.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[12] I. Megyeri, I. Hegedűs, and M. Jelasity, "Attacking model sets with adversarial examples," in *Proceedings of the 28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, Bruges, Belgium, 2020.

[13] Z. Yao, A. Gholami, P. Xu, K. Keutzer, and M. W. Mahoney, "Trust region based adversarial attack on neural networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019, pp. 11 342–11 351.

[14] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.

[15] R. Breitling, P. Armengaud, A. Amtmann, and P. Herzyk, "Rank products: A simple, yet powerful, new method to detect differentially regulated genes in replicated microarray experiments," *FEBS letters*, vol. 573, pp. 83–92, 09 2004.

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2018, pp. 4510–4520.

[17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[18] B. Zoph, V. Vasudevan, J. Shlens, and Q. Le, "Learning transferable architectures for scalable image recognition," 07 2017.

[19] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.