Towards Reverse-Engineering Black-Box Neural Networks

Seong Joon Oh Max Augustin Bernt Schiele Mario Fritz
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
{joon,maxaug,schiele,mfritz}@mpi-inf.mpg.de

1. Introduction

Many deployed learned models are black boxes: given input, returns output. Internal information about the model, such as the architecture, optimisation procedure, or training data, is not disclosed explicitly as it might contain proprietary information or make the system more vulnerable. This work shows that such attributes of neural networks can be exposed from a sequence of queries. This has multiple implications. On the one hand, our work exposes the vulnerability of black-box neural networks to different types of attacks – we show that the revealed internal information helps generate more effective adversarial examples against the black box model. On the other hand, this technique can be used for better protection of private content from automatic recognition models using adversarial examples. Our paper suggests that it is actually hard to draw a line between white box and black box models.

We consider various types of "model attributes" to reveal: see table 1 for a summary. We approach the problem as a standard supervised learning task *applied over models*. First, collect a diverse set of white-box models ("meta-training set") that are expected to be similar to the target black box at least to a certain extent. Then, over the collected meta-training set, train another model ("metamodel") that takes a model as input and returns the model attributes as output. Importantly, since we want to predict attributes at test time for black-box models, the only information available for attribute prediction is the query input-output pairs. As we will see in the experiments, such input-output pairs allow to predict model attributes surprisingly well.

2. Metamodels

For the method for collecting the meta-training set, see the main paper. We only introduce the architecture and training/testing procedures for the metamodels. A metamodel submits n query inputs $\left[x^i\right]_{i=1}^n$ to a black box model g; the metamodel observes the corresponding model outputs $\left[g(x^i)\right]_{i=1}^n$ and predicts model attributes. The metamodel is trained over meta-training models f in the training set ($f \sim \mathcal{F}$). We propose three novel `kennen`[1] metamodels. See figure 1 for an overview. We assume an MNIST classifier for the sake of clarity.

Table 1: MNIST classifier attributes that can be revealed via black-box access.

	Code	Attribute	Values
Architecture	act	Activation	ReLU, PReLU, ELU, Tanh
	drop	Dropout	Yes, No
	pool	Max pooling	Yes, No
	ks	Conv ker. size	3, 5
	#conv	#Conv layers	2, 3, 4
	#fc	#FC layers	2, 3, 4
	#par	*#Parameters*	2^{14}, $\cdots$, 2^{21}
	ens	Ensemble	Yes, No
Opt.	alg	Algorithm	SGD, ADAM, RMSprop
	bs	Batch size	64, 128, 256
Data	split	Data split	All_0, $\text{Half}_{0/1}$, $\text{Quarter}_{0/1/2/3}$
	size	*Data size*	All, Half, Quarter

`kennen-o`: reason over output `kennen-o` first selects a fixed set of queries $[x^i]_{i=1\cdots n}$ from a dataset. Both during training and testing, always these queries are submitted. `kennen-o` learns a classifier m_θ to map from the order-sensitively concatenated n query outputs, $[f(x^i)]_{i=1\cdots n}$ ($n \times 10$ dim for MNIST), to the simultaneous prediction of 12 attributes in f. The training objective is:

$$\min_\theta \mathop{\mathbb{E}}_{f\sim\mathcal{F}} \left[\sum_{a=1}^{12} \mathcal{L}\left(m_\theta^a\left(\left[f(x^i)\right]_{i=1}^n\right), y^a\right) \right] \qquad (1)$$

where y^a is the ground truth label of attribute a, and $\mathcal{L}$ is the cross-entropy loss. With the learned parameter $\tilde{\theta}$, $m_{\tilde{\theta}}^a\left(\left[g(x^i)\right]_{i=1}^n\right)$ gives the prediction of attribute a for the black box g.

`kennen-i`: craft input `kennen-i` approaches the problem from a completely new point of view. Over multiple training set models (white boxes), `kennen-i` crafts

[1] *kennen* means "to know" in German, and "to dig out" in Korean.

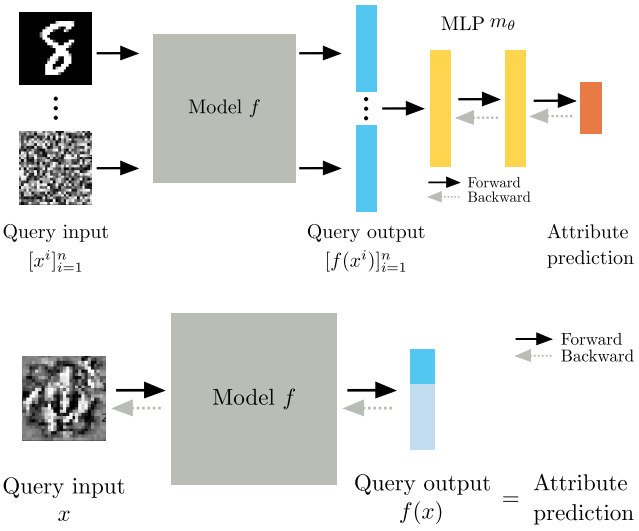

Figure 1: Training procedure for metamodels `kennen-o` (top) and `kennen-i` (bottom).

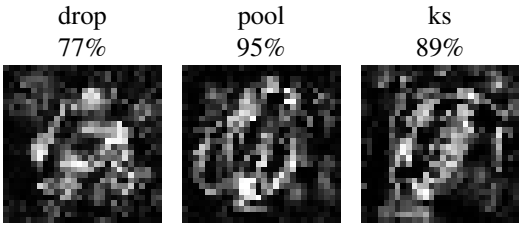

Figure 2: Inputs designed to extract internal details from MNIST digit classifiers. E.g. feeding the middle image reveals the existence of a max-pooling layer with 95% chance.

a *single* query input $\tilde{x}$ that is designed to expose inner secrets of the training set models. This crafted input turns out to generalise very well to unseen black-box models, in the sense that it also reveals the secrets of the unseen black box. More specifically, we design a query input that forces an MNIST digit classifier to predict "1" if the classifier has the attribute A, and 0 otherwise, for example. In other words, the crafted input re-purposes a digit classifier into a model attribute classifier. See figure 1 and the learning objective below for the full detail.

$$\min_{x:\,\text{image}} \mathop{\mathbb{E}}_{f\sim\mathcal{F}} \left[\mathcal{L}\left(f(x), y^a\right)\right] \tag{2}$$

where $f(x)$ is the 10-dimensional output of the digit classifier f. The condition x : image ensures the input stays a valid image $x \in [0,1]^D$ with image dimension D. The loss $\mathcal{L}$, together with the attribute label y^a of f, guides the digit prediction $f(x)$ to reveal the attribute a instead. With the learned query input $\tilde{x}$, the attribute for the black box g is predicted by computing $g(\tilde{x})$. In particular, we only need black-box access to g. Example crafted inputs is shown in figure 2.

Table 2: Comparison of metamodel methods. See table 1 for the full names of attributes. Ch.=Chance level.

	architecture								optim		data		
	act	drop	pool	ks	#conv	#fc	#par	ens	alg	bs	size	split	avg
Ch.	25	50	50	50	33	33	13	50	33	33	33	14	35
`o`	81	95	95	85	67	77	42	54	72	50	74	90	73
`i`	44	77	95	89	55	41	32	47	46	37	43	29	53
`io`	**88**	**96**	**100**	**98**	**80**	**80**	**45**	**60**	**79**	**54**	**85**	**96**	**80**

`kennen-io`: combined approach This method combines the previous two approaches.

3. Teaser Results on MNIST Digit Classifiers

See table 2 for the black-box attribute prediction performances of our three metamodels, `kennen-o/i/io` trained on $5,000$ meta-training classifiers with $n = 100$ queries with probability output (except for `kennen-i` which only uses one query). `kennen-o` already performs far above the random chance in predicting 12 diverse attributes (73% versus 35% on average); neural network output indeed contains rich information about the black box. In particular, the presence of dropout (95%) or max-pooling (95%) has been predicted with high precision. It is surprising that optimisation details like algorithm (72%) and batch size (59%) can also be predicted well above the random chance (33% for both). We observe that the training data attributes are also predicted with high accuracy (72% and 90% for size and split).

`kennen-i` has a relatively low performance (average 53%), but `kennen-i` relies on a cheap resource: 1 query with single-label output. `kennen-i` is also performant at predicting the kernel size (89%) and pooling (95%), attributes that are closely linked to spatial structure of the input. We conjecture `kennen-i` is relatively effective for such attributes. `kennen-io` is superior to `kennen-o/i` for all the attributes with average accuracy 80%.

4. More in the ICLR'18 paper!

The main paper[2] contains more surprises. (1) Extrapolation works: the proposed metamodels still predicts attributes reliably well even if the meta-training models are significantly different from the test black box of interest (e.g. different depth). (2) It is also possible to extract internal information from top-k or even single-label outputs. (3) Exposed internal information makes the model more vulnerable to adversarial examples (tested over ImageNet classifiers). The code for generating the meta-training models over MNIST and for training the `kennen` variants are available on github[3].

[2] https://openreview.net/forum?id=BydjJte0-
[3] https://github.com/coallaoh/WhitenBlackBox