Selected ICLR 2017 Papers

Agnieszka Pocha

Jagiellonian University

19.05.2017, Kraków

Designing Neural Network Architectures using Reinforcement Learning

Bowen Baker, Otkrist Gupta, Nikhil Naik, Ramesh Raskar

Abstract

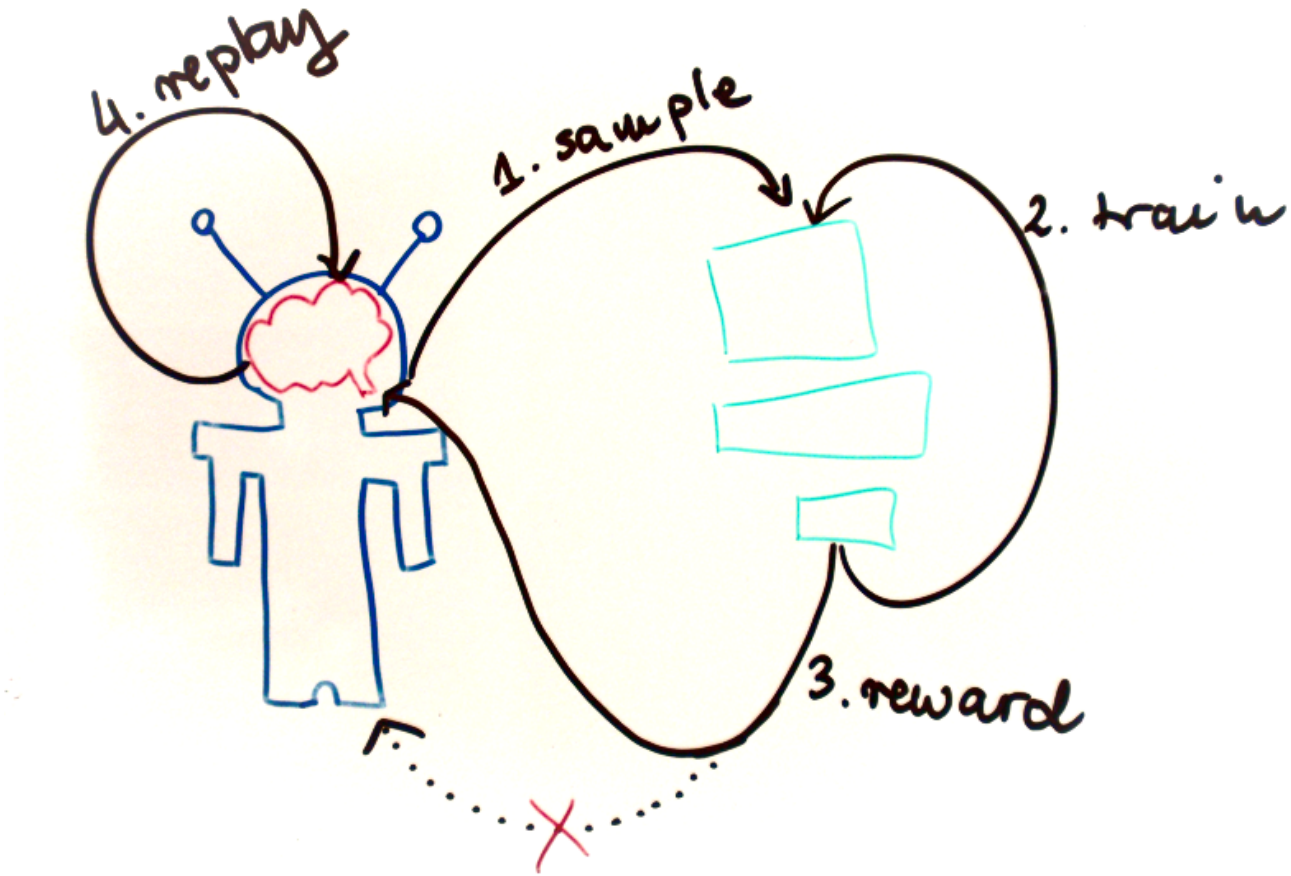
Designing NN architectures is slow and laborious. We would like to have an automatic and successful method to do it for us. Authors introduce

MetaQNN - a **reinforcement learning method** based on **Q-learnig** algorithm that finds highly performing architectures. Their models:

- beat on CIFAR, MNIST & SVHN best performing models that share similar architecture
- compare well with state-of-the-art models
- are suitable for transfer learning

Moreover, authors show that their method is stable.

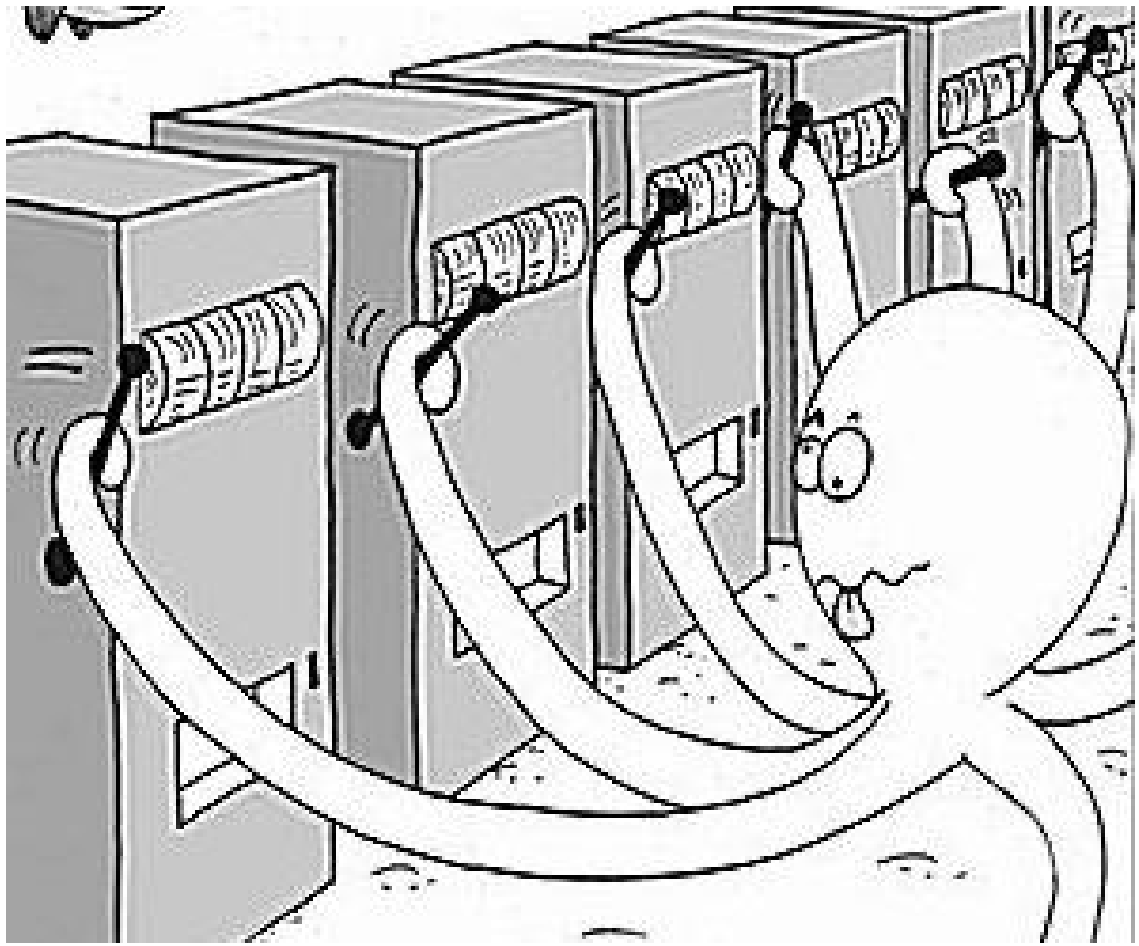
Shortly



Learning agent sequentially chooses CNN layers using **Q-learnig** with an ϵ-**greedy exploration strategy** and **experience replay**.

ϵ-greedy exploration strategy

Choose what you think is the best option with probability $1 - \epsilon$ and choose a random action with probability ϵ.



http://research.microsoft.com/en-us/projects/bandits/

Q-learnig

Let $\mathcal{S}$ - state space, $\mathcal{U}$- action space, $\mathcal{U}(s_i) \in \mathcal{U}$ - actions possible while in state s_i.

In an environment with stochastic transitions, an agent in state s_i taking some action $u \in \mathcal{U}$ will **transition** to state s_j with probability $p_{s'|s,u}(s_j|s_i, u)$ which may be unknown to the agent.

At each timestep t, the agent is given a **reward** r_t, dependent on the transition from state s to s' and action u. The reward may also be stochastic according to a distribution $p_{r|s',s,u}$.

The agent's **goal** is to maximize the **total expected reward** over all possible trajectories, i.e. $max_{\tau_i \in \tau} R_{\tau_i}$, where the total expected reward for a trajectory τ_i is

$$R_{\tau_i} = \Sigma_{(s,u,s') \in \tau_i} \mathbb{E}_{r|s,u,s'}[r|s, u, s']$$

Q-learnig

Let $\mathcal{S}$ - state space, $\mathcal{U}$- action space, $\mathcal{U}(s_i) \in \mathcal{U}$ - actions possible while in state s.

$$R_{\tau_i} = \Sigma_{(s,u,s') \in \tau_i} \mathbb{E}_{r|s,u,s'}[r|s,u,s']$$

The number of possible trajectories makes the problem untractable, therefore we define the maximization problem recursively in terms of subproblems as follows: for any state $s_i \in S$ and subsequent action $u \in \mathcal{U}(s_i)$, we define the **maximum total expected reward** to be $Q^*(s_i, u)$. The recursive maximization equation, which is known as **Bellman's Equation**, can be written as:

$$Q^*(s_i, u) = \mathbb{E}_{s_j|s_i,u}[\mathbb{E}_{r|s_i,u,s_j}[r|s_i,u,s_j] + \gamma max_{u' \in \mathcal{U}(s_j)} Q^*(s_j, u')]$$

We usually cannot solve it analytically but we can define an **iterative update**.

Q-learning

- **model-free**
- **off-policy**
- two parameters:
  - α - **learning rate**
  - λ - **discount factor** (weight given to short term rewards over future rewards)

Here:

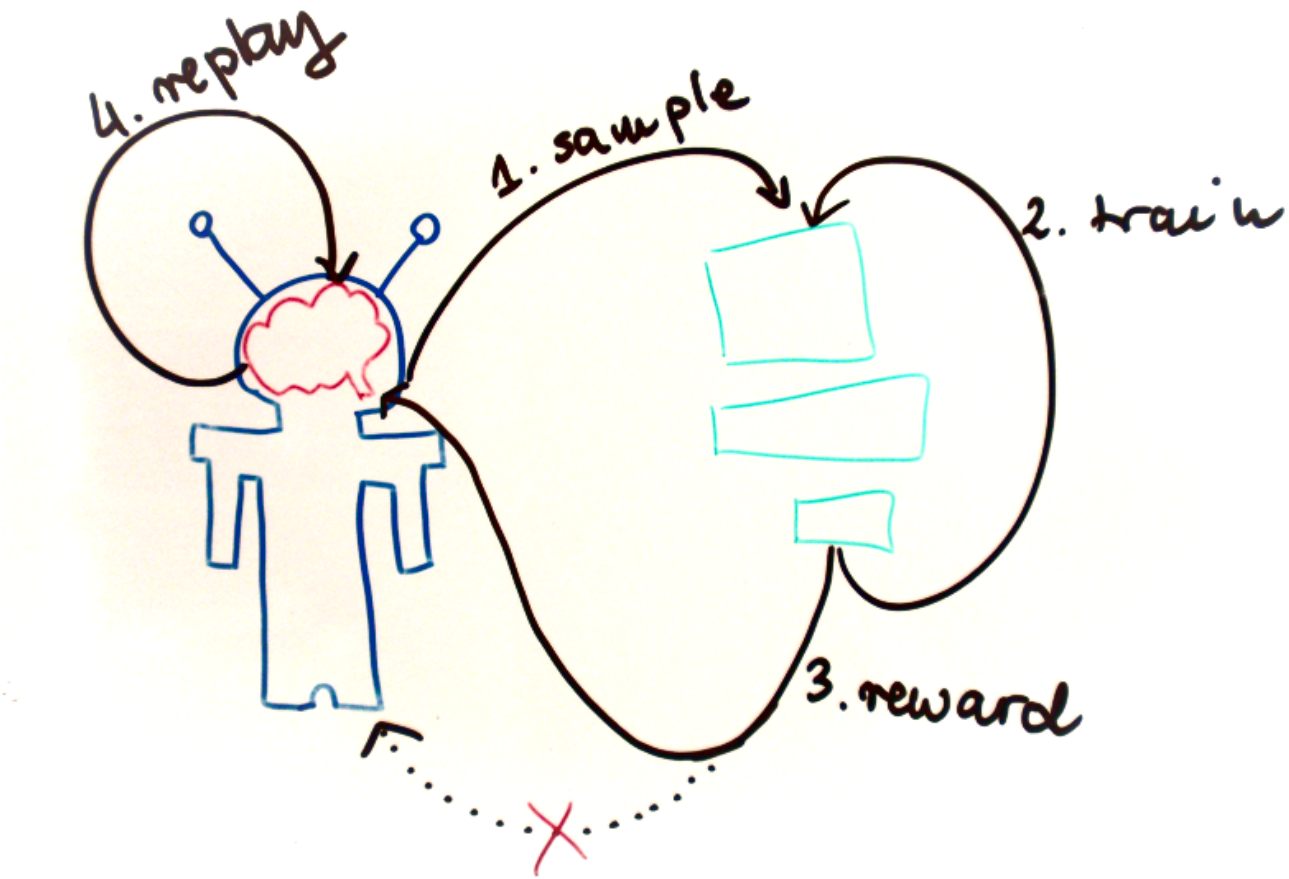
- **action** - choosing next layer along with its parameters (size, stride...)
- **state** - the current state of the network topology
- **reward** - performance on validation set (5K samples, with unchanged class distribution)

Experience replay

Use single experience multiple times for an iterative update.

https://www.shutterstock.com/image-photo/empty-toilet-paper-roll-recycled-seedling-134967662?src=ijY4nrNlx43dUDT2n_Cjsw-1-4

Shortly (again)



Learning agent sequentially chooses CNN layers using **Q-learnig** with an ϵ-**greedy exploration strategy** and **experience replay**.

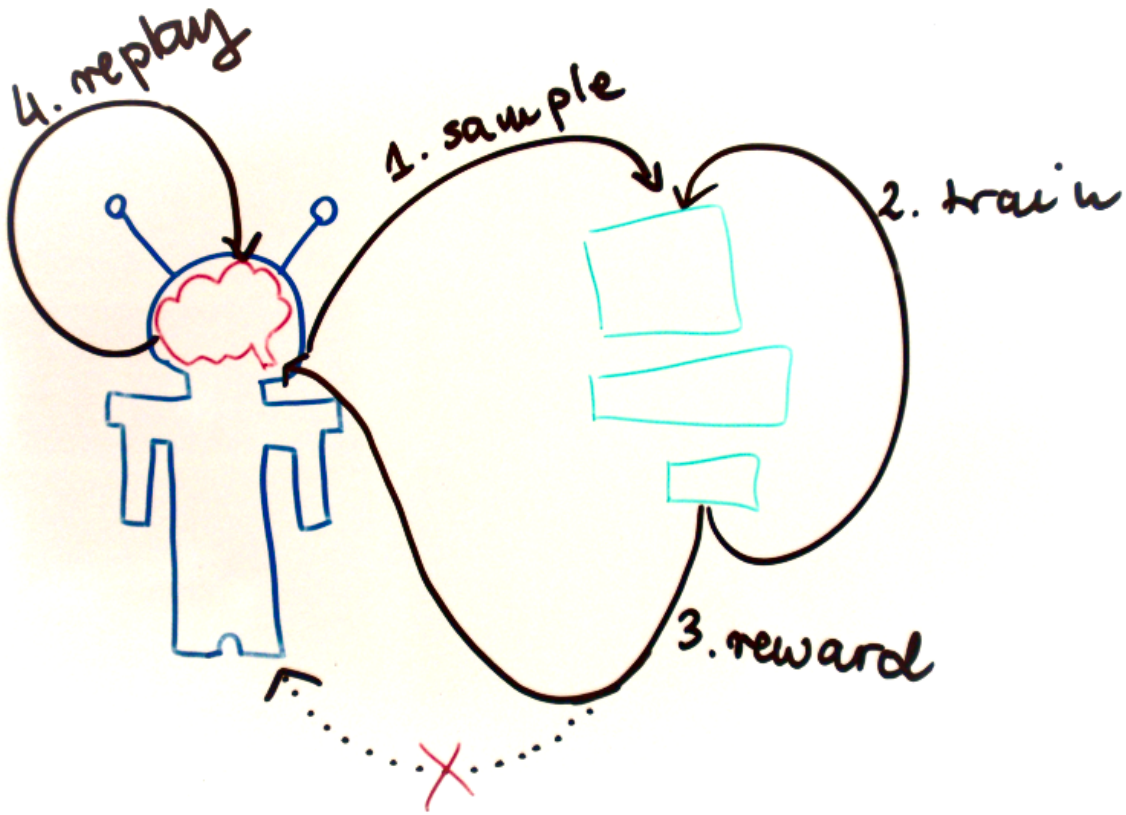
Training models

Models were trained in an aggresive fashion (stop this violence!) what made the process less costly.

- dropout every 2 layers
- 20 epochs
- Adam
- batch size 128
- learning rate 0.001 reduced by a factor of 0.2 every 5 epochs
- Xavier initialization (Glorot & Bengio 2010)

Time: 8-10 days for each dataset on 10 NVIDIA GPUs (GMUM needs more GPUs).

Again same picture but now we should understand it completely



Learning agent sequentially chooses CNN layers using **Q-learnig** with an ϵ-**greedy exploration strategy** and **experience replay**.

Experiments

It grows! It means that the architectures chosen greedily/later (exploitation) are better than those randomly sampled (exploration) - the agent has learned something.

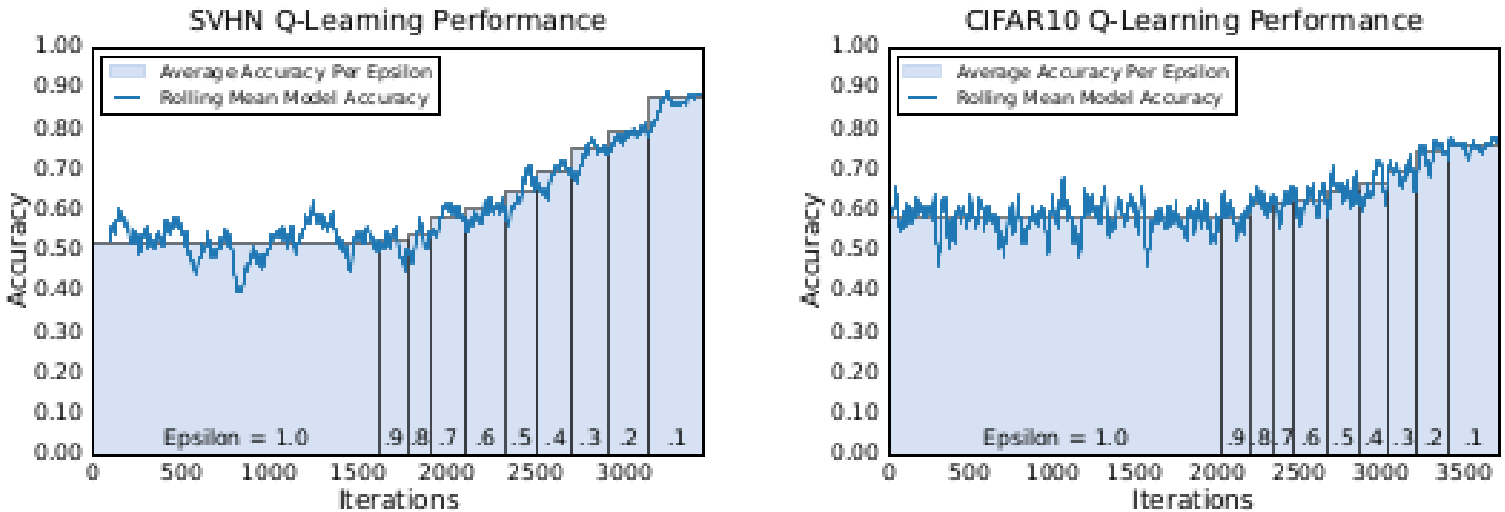
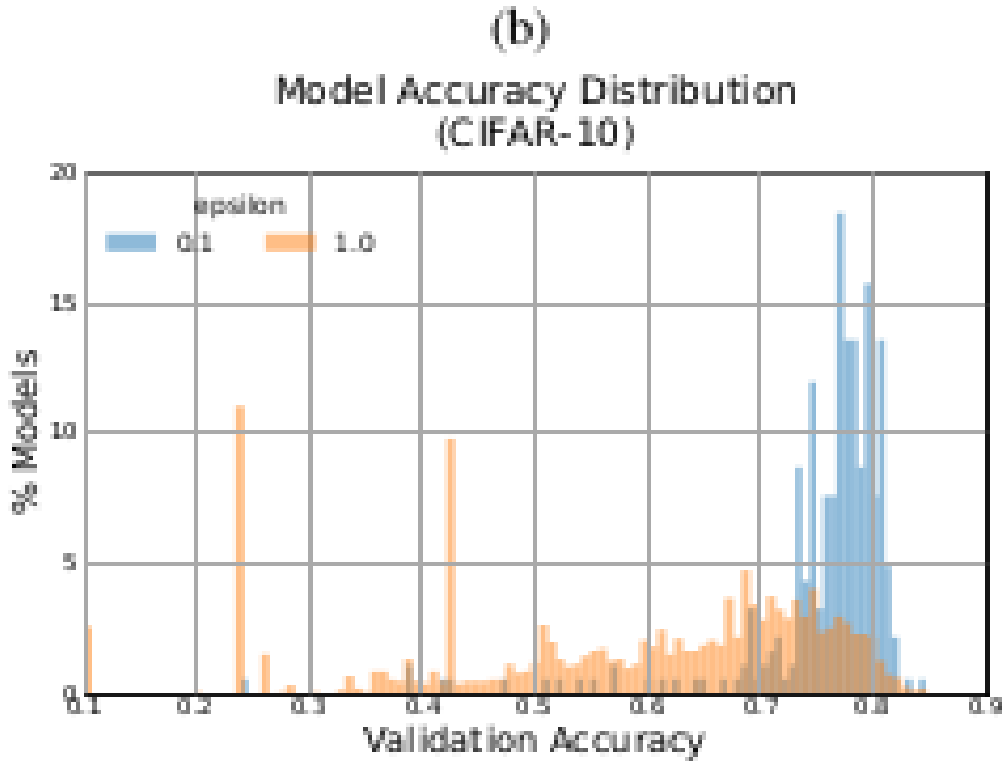


Figure 3: ***Q*-Learning Performance.** In the plots, the blue line shows a rolling mean of model accuracy versus iteration, where in each iteration of the algorithm the agent is sampling a model. Each bar (in light blue) marks the average accuracy over all models that were sampled during the exploration phase with the labeled ϵ. As ϵ decreases, the average accuracy goes up, demonstrating that the agent learns to select better-performing CNN architectures.

Experiments

It learns! It means that the architectures chosen greedily/later (exploitation) are better than those randomly sampled (exploration) - the agent has learned something.



Results

After short training all those architectures 10 best were chosen and well tuned. Finally, 5 best were chosen to become ensemble.

- beat on CIFAR, MNIST & SVHN best performing models that share similar architecture

Method	CIFAR-10	SVHN	MNIST	CIFAR-100
Maxout (Goodfellow et al., 2013)	9.38	2.47	0.45	38.57
NIN (Lin et al., 2013)	8.81	2.35	0.47	35.68
FitNet (Romero et al., 2014)	8.39	2.42	0.51	35.04
HighWay (Srivastava et al., 2015)	7.72	-	-	-
VGGnet (Simonyan & Zisserman, 2014)	7.25	-	-	-
All-CNN (Springenberg et al., 2014)	7.25	-	-	33.71
MetaQNN (ensemble)	7.32	**2.06**	**0.32**	-
MetaQNN (top model)	**6.92**	2.28	0.44	**27.14***

Results

- compare well with state-of-the-art models

Method	CIFAR-10	SVHN	MNIST	CIFAR-100
DropConnect (Wan et al., 2013)	9.32	1.94	0.57	-
DSN (Lee et al., 2015)	8.22	1.92	0.39	34.57
R-CNN (Liang & Hu, 2015)	7.72	1.77	**0.31**	31.75
MetaQNN (ensemble)	7.32	2.06	0.32	-
MetaQNN (top model)	6.92	2.28	0.44	27.14*
Resnet(110) (He et al., 2015)	6.61	-	-	-
Resnet(1001) (He et al., 2016)	**4.62**	-	-	**22.71**
ELU (Clevert et al., 2015)	6.55	-	-	24.28
Tree+Max-Avg (Lee et al., 2016)	6.05	**1.69**	**0.31**	32.37

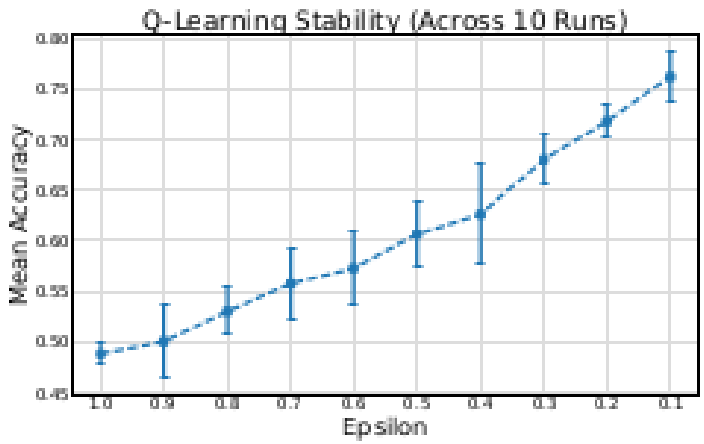
- are suitable for transfer learning

Dataset	CIFAR-100	SVHN	MNIST
Training from scratch	27.14	2.48	0.80
Finetuning	34.93	4.00	0.81
State-of-the-art	24.28 (Clevert et al., 2015)	1.69 (Lee et al., 2016)	0.31 (Lee et al., 2016)

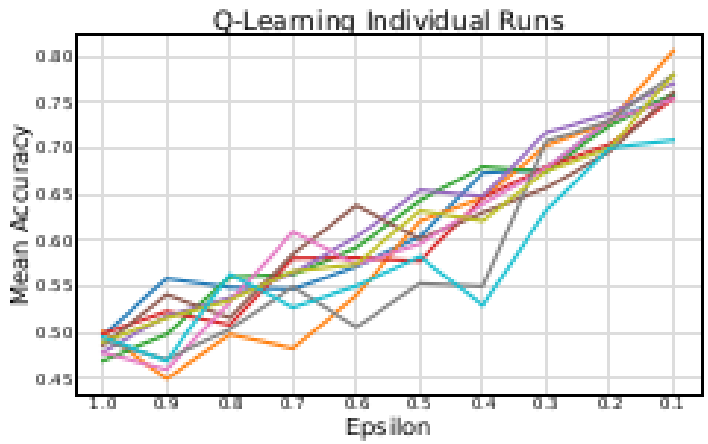
Table 5: **Prediction Error** for the top MetaQNN (CIFAR-10) model trained for other tasks. Fine-tuning refers to initializing training with the weights found for the optimal CIFAR-10 model.

Results

- stability



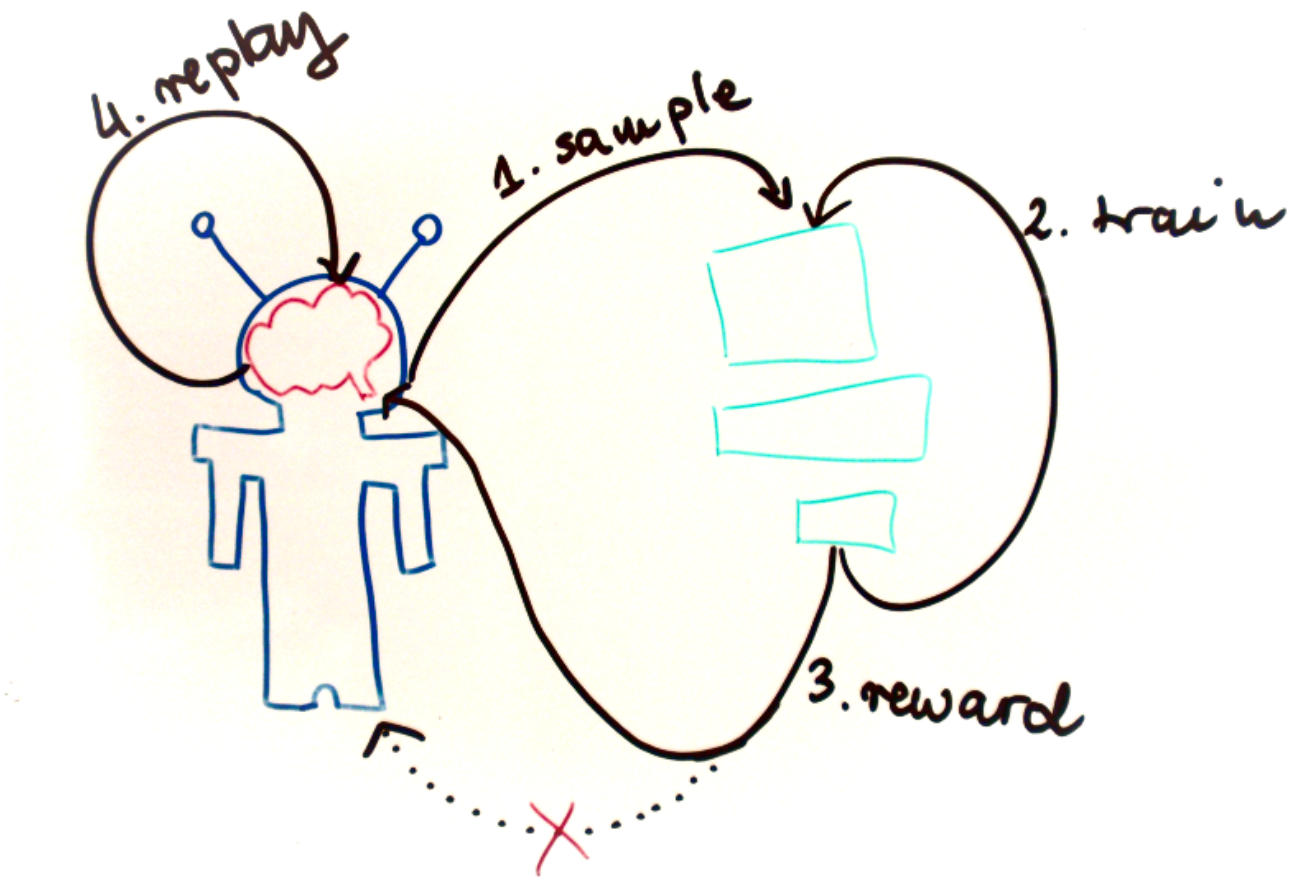
(a)



(b)

Figure A3: Figure A3a shows the mean model accuracy and standard deviation at each ϵ over 10 independent runs of the Q-learning procedure on 10% of the SVHN dataset. Figure A3b shows the mean model accuracy at each ϵ for each independent experiment. Despite some variance due to a randomized exploration strategy, each independent run successfully improves architecture performance.

My problem with this paper



Late architectures (best performing) might not be chosen for replay i.e. not used for training at all while early architectures (randomly performing) will be overrepresented.

My problem with this paper

Algorithm 1 Q-learning For CNN Topologies

Initialize:
replay_memory $\leftarrow [\,]$
$Q \leftarrow \{(s, u)\ \forall s \in \mathcal{S}, u \in \mathcal{U}(s)\ :\ 0.5\}$
for episode = 1 to M **do**
$S, U \leftarrow$ SAMPLE_NEW_NETWORK(ϵ, Q)
accuracy $\leftarrow$ TRAIN(S)
replay_memory.append((S, U, accuracy))
for memory = 1 to K **do**
S_{SAMPLE}, U_{SAMPLE}, accuracy$_{SAMPLE} \leftarrow$ Uniform{replay_memory}
$Q \leftarrow$ UPDATE_Q_VALUES(Q, S_{SAMPLE}, U_{SAMPLE}, accuracy$_{SAMPLE}$)
end for
end for

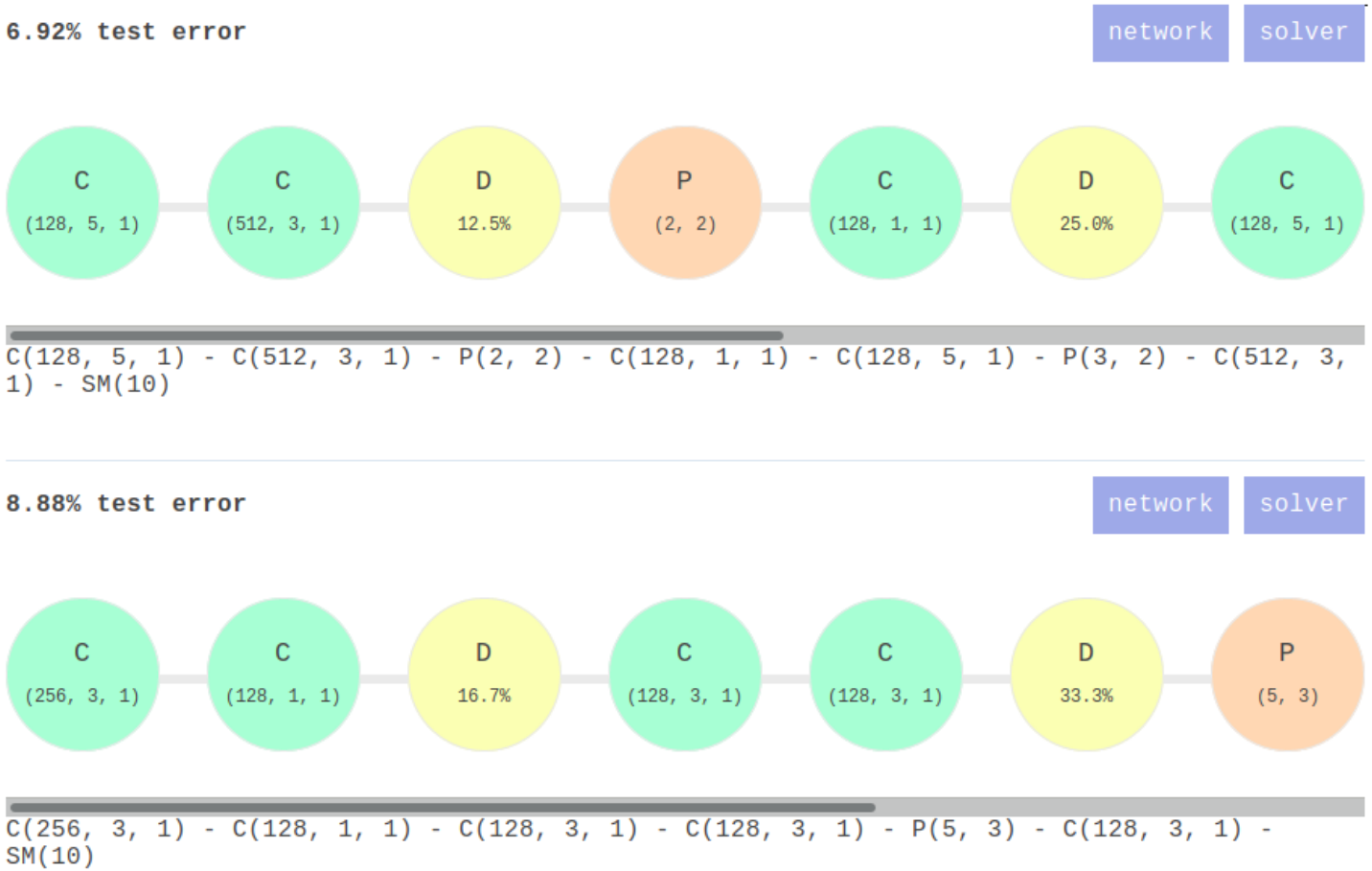
Late architectures (best performing) might not be chosen for replay i.e. not used for training at all, while early architectures (randomly performing) will be overrepresented.

WHY?

Important note

"While we report results for image classification problems, our method could be applied to different problem settings, including supervised (e.g., classification, regression) and unsupervised (e.g., autoencoders)"

More



- arXiv 1611.02167 https://arxiv.org/abs/1611.02167
- GitHub https://bowenbaker.github.io/metaqnn/

Adversarial Examples in the Physical World

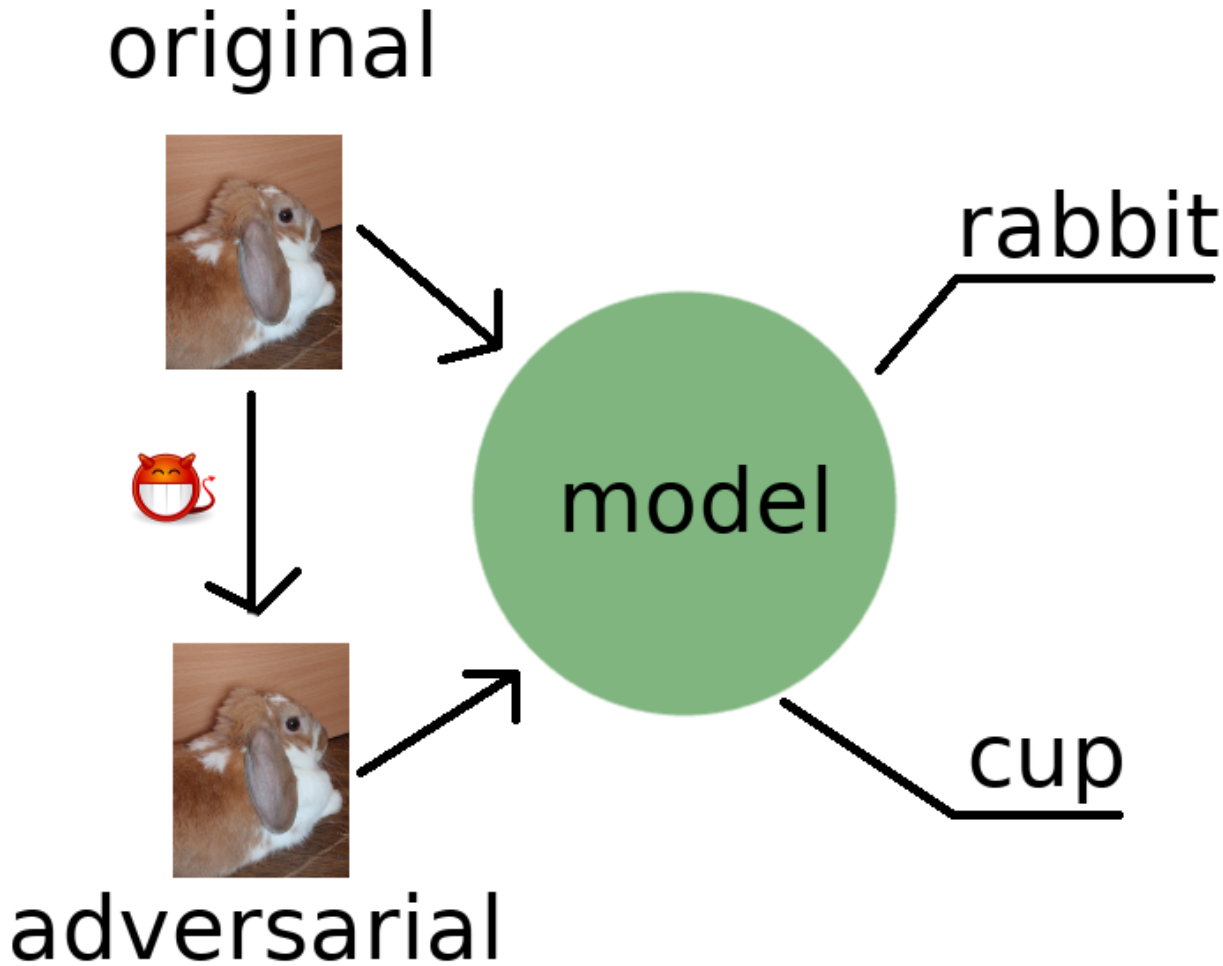
Alexey Kurakin, Ian J. Goodfellow, Samy Bengio

Abstract

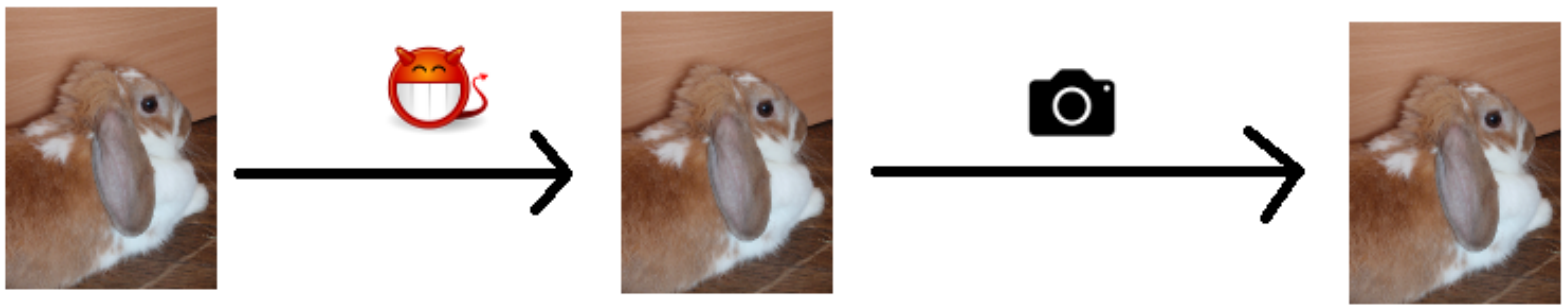
An adversarial example is a sample of input data which has been modified very slightly in a way that is intended to cause a machine learning classifier to missclassify it. Adversarial examples pose security concerns because they could be used to perform attack on machine learning systems.

1. Authors found that a large fraction of adversarial examples generated for the original model remain missclassified even when perceived through a camera or altered with another transformation.
2. They also demonstrated that the physical adversarial sample constructed for one model would fool another model.

Adversarial Examples

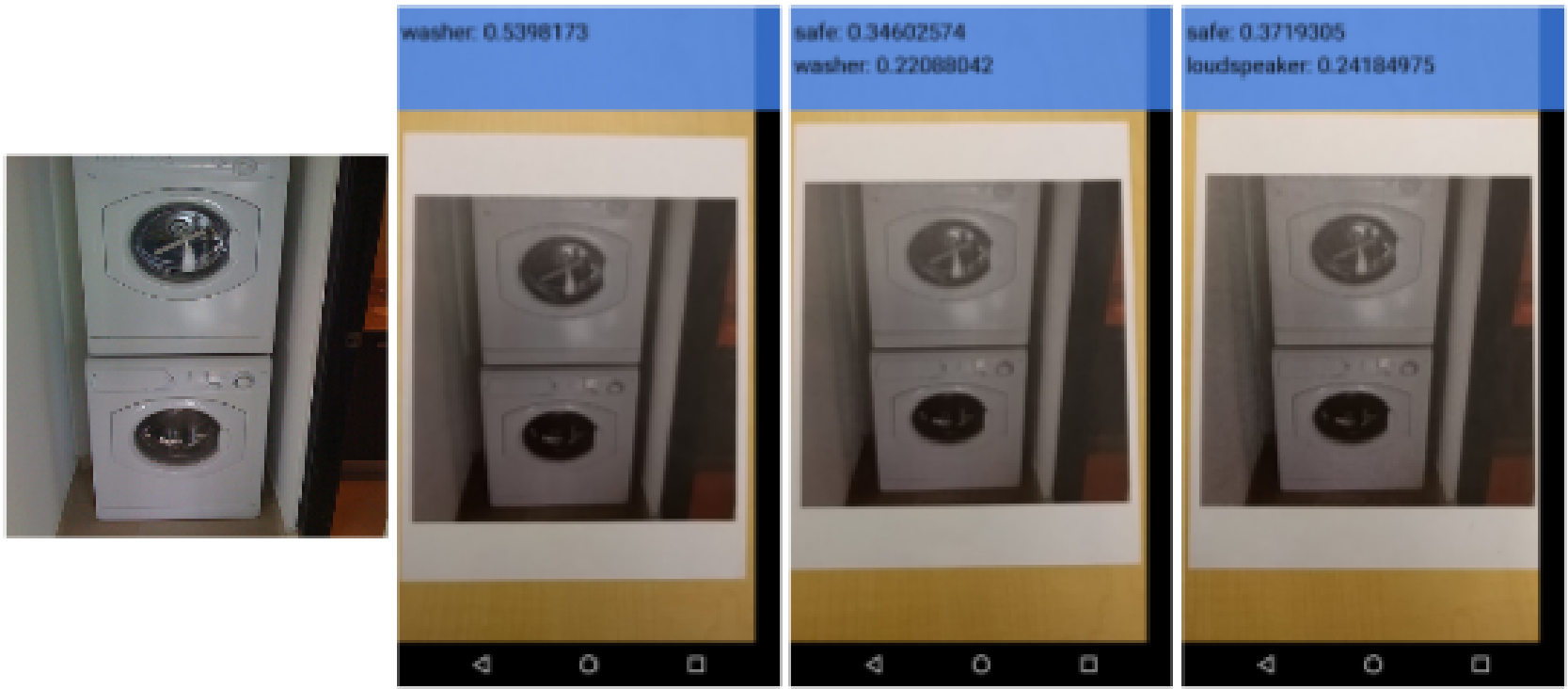


Setup



?

Actual outcome



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

Methods for generating adversarial examples

- **Fast method**

$$\mathbf{X}^{adv} = \mathbf{X} + \epsilon sign(\nabla_X J(\mathbf{X}, y_{true}))$$

- **Basic iterative method**

$$\mathbf{X}_0^{adv} = \mathbf{X}, \mathbf{X}_{N+1}^{adv} = Clip_{X,\epsilon}\{\mathbf{X}_N^{adv} + \alpha sign(\nabla_X J(\mathbf{X}_N^{adv}, y_{true}))\}$$

- **Iterative least-likely class method**

$$\mathbf{X}_0^{adv} = \mathbf{X}, \mathbf{X}_{N+1}^{adv} = Clip_{X,\epsilon}\{\mathbf{X}_N^{adv} - \alpha sign(\nabla_X J(\mathbf{X}_N^{adv}, y_{LL}))\}$$

where $J(\cdot,\cdot)$ is a cross-entropy cost function of the neural network, $Clip_{X,\epsilon}\{\mathbf{X}'\}$ is a function which performs per-pixel clipping of the image $\mathbf{X}'$ so that the result will be in the neighbourhood of the source image $\mathbf{X}$, y_{LL} is the least likely class.

Methods for generating adversarial examples

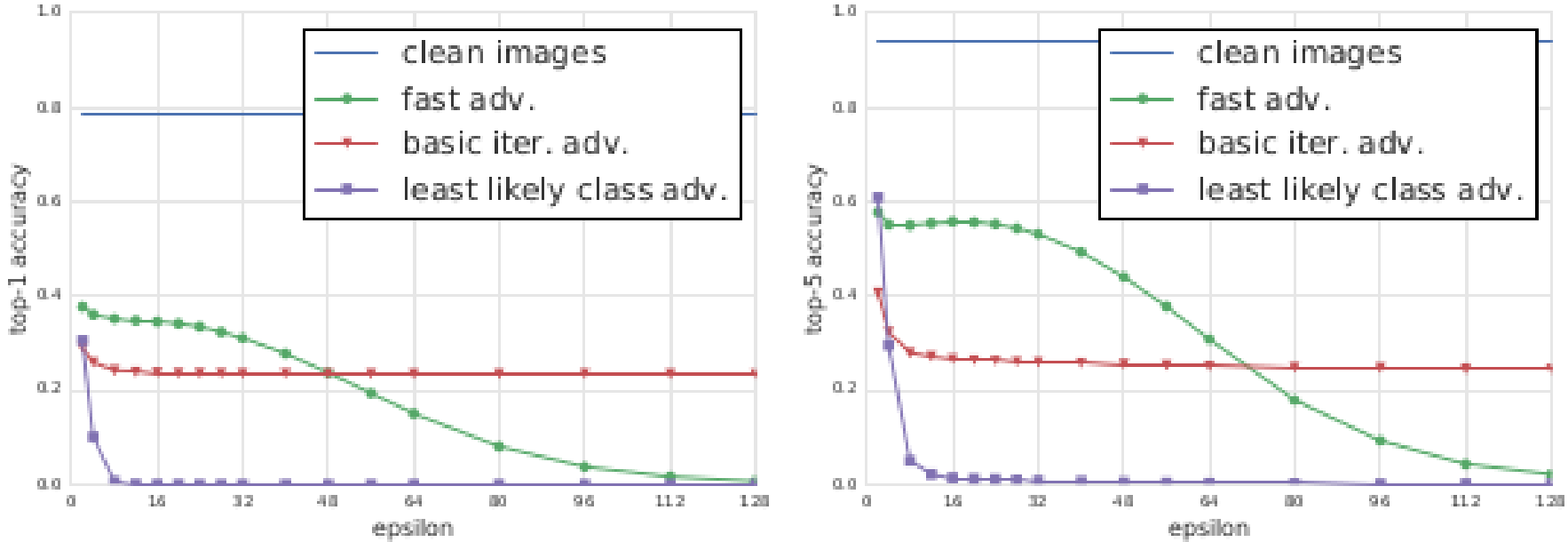


Figure 2: Top-1 and top-5 accuracy of Inception v3 under attack by different adversarial methods and different ϵ compared to "clean images" — unmodified images from the dataset. The accuracy was computed on all $50,000$ validation images from the ImageNet dataset. In these experiments ϵ varies from 2 to 128.

None of these methods guarantees that generated image will be misclassified.

What is the difference between those methods?

Clean image

"Fast"; L_∞ distance to clean image = 32

"Basic iter."; L_∞ distance to clean image = 32

"L.l. class"; L_∞ distance to clean image = 28

Figure 4: Comparison of different adversarial methods with $\epsilon = 32$. Perturbations generated by iterative methods are finer compared to the fast method. Also iterative methods do not always select a point on the border of ϵ-neighbourhood as an adversarial image.

Okay, but what doeas it mean: epsilon = 32?

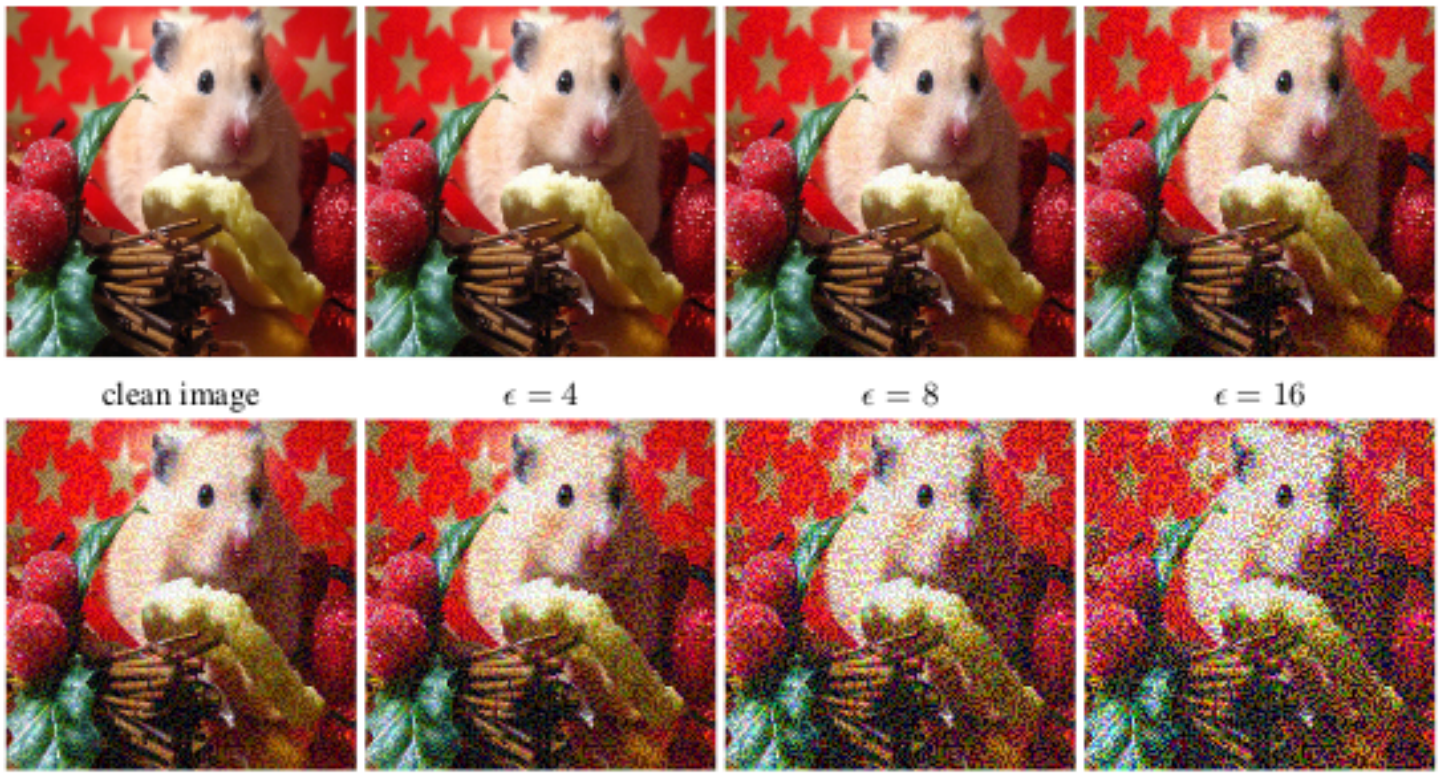


Figure 5: Comparison of images resulting from an adversarial pertubation using the "fast" method with different size of perturbation ϵ. The top image is a "washer" while the bottom one is a "hamster". In both cases clean images are classified correctly and adversarial images are misclassified for all considered ϵ.

Experiments

Destruction rate

Let n be number of images used to compute the destruction rate, $\mathbf{X}^k$ an image from dataset, y^k_{true} - true class of $k-th$ image, $\mathbf{X}^k_{adv}$ - the corresponding adversarial image, $T(\cdot)$ - an arbitrary image transformation, $C(\mathbf{X}, y)$ - an indicator function:

$$C(\mathbf{X}, y) = \begin{cases} 1 & \text{if image } \mathbf{X} \text{ is classified as y,} \\ 0 & \text{otherwise ,} \end{cases}$$

Then, the destruction rate is defined as:

$$d = \frac{\sum_{k=1}^{n} C(\mathbf{X}^k, y^k_{true})\overline{C(\mathbf{X}^k_{adv}, y^k_{true})}C(T(\mathbf{X}^k_{adv}), y^k_{true})}{\sum_{k=1}^{n} C(\mathbf{X}^k, y^k_{true})\overline{C(\mathbf{X}^k_{adv}, y^k_{true})}}$$

where $\overline{C(\mathbf{X}, y)}$ is a binary negation.

Experiments

Average case - images used to calculate measures were chosen randomly.

Prefiltered case (aggresive attack (stop this violence!)) - images used to calculate measures were chosen in such a way that all clean images were classified correctly and all adversarial images were classified incorrectly.

Results

1. Authors found that a large fraction of adversarial examples generated for the original model remain missclassified even when perceived through a camera or altered with another transformation.
2. They also demonstrated that the physical adversarial sample constructed for one model would fool another model.

Results

Average case accuracy

Table 1: Accuracy on photos of adversarial images in the average case (randomly chosen images).

Adversarial method	Photos				Source images			
	Clean images		Adv. images		Clean images		Adv. images	
	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5
fast $\epsilon = 16$	79.8%	91.9%	36.4%	67.7%	85.3%	94.1%	36.3%	58.8%
fast $\epsilon = 8$	70.6%	93.1%	49.0%	73.5%	77.5%	97.1%	30.4%	57.8%
fast $\epsilon = 4$	72.5%	90.2%	52.9%	79.4%	77.5%	94.1%	33.3%	51.0%
fast $\epsilon = 2$	65.7%	85.9%	54.5%	78.8%	71.6%	93.1%	35.3%	53.9%
iter. basic $\epsilon = 16$	72.9%	89.6%	49.0%	75.0%	81.4%	95.1%	28.4%	31.4%
iter. basic $\epsilon = 8$	72.5%	93.1%	51.0%	87.3%	73.5%	93.1%	26.5%	31.4%
iter. basic $\epsilon = 4$	63.7%	87.3%	48.0%	80.4%	74.5%	92.2%	12.7%	24.5%
iter. basic $\epsilon = 2$	70.7%	87.9%	62.6%	86.9%	74.5%	96.1%	28.4%	41.2%
l.l. class $\epsilon = 16$	71.1%	90.0%	60.0%	83.3%	79.4%	96.1%	1.0%	1.0%
l.l. class $\epsilon = 8$	76.5%	94.1%	69.6%	92.2%	78.4%	98.0%	0.0%	6.9%
l.l. class $\epsilon = 4$	76.8%	86.9%	75.8%	85.9%	80.4%	90.2%	9.8%	24.5%
l.l. class $\epsilon = 2$	71.6%	87.3%	68.6%	89.2%	75.5%	92.2%	20.6%	44.1%

Results

Prefiltered case accuracy

Table 2: Accuracy on photos of adversarial images in the prefiltered case (clean image correctly classified, adversarial image confidently incorrectly classified in digital form being being printed and photographed).

Adversarial method	Photos				Source images			
	Clean images		Adv. images		Clean images		Adv. images	
	top-1	top-5	top-1	top-5	top-1	top-5	top-1	top-5
fast $\epsilon = 16$	81.8%	97.0%	5.1%	39.4%	100.0%	100.0%	0.0%	0.0%
fast $\epsilon = 8$	77.1%	95.8%	14.6%	70.8%	100.0%	100.0%	0.0%	0.0%
fast $\epsilon = 4$	81.4%	100.0%	32.4%	91.2%	100.0%	100.0%	0.0%	0.0%
fast $\epsilon = 2$	88.9%	99.0%	49.5%	91.9%	100.0%	100.0%	0.0%	0.0%
iter. basic $\epsilon = 16$	93.3%	97.8%	60.0%	87.8%	100.0%	100.0%	0.0%	0.0%
iter. basic $\epsilon = 8$	89.2%	98.0%	64.7%	91.2%	100.0%	100.0%	0.0%	0.0%
iter. basic $\epsilon = 4$	92.2%	97.1%	77.5%	94.1%	100.0%	100.0%	0.0%	0.0%
iter. basic $\epsilon = 2$	93.9%	97.0%	80.8%	97.0%	100.0%	100.0%	0.0%	1.0%
l.l. class $\epsilon = 16$	95.8%	100.0%	87.5%	97.9%	100.0%	100.0%	0.0%	0.0%
l.l. class $\epsilon = 8$	96.0%	100.0%	88.9%	97.0%	100.0%	100.0%	0.0%	0.0%
l.l. class $\epsilon = 4$	93.9%	100.0%	91.9%	98.0%	100.0%	100.0%	0.0%	0.0%
l.l. class $\epsilon = 2$	92.2%	99.0%	93.1%	98.0%	100.0%	100.0%	0.0%	0.0%

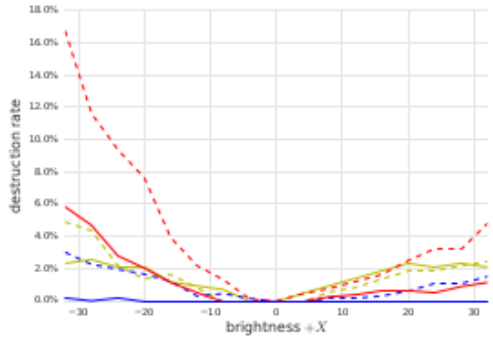
Results

Destruction rate

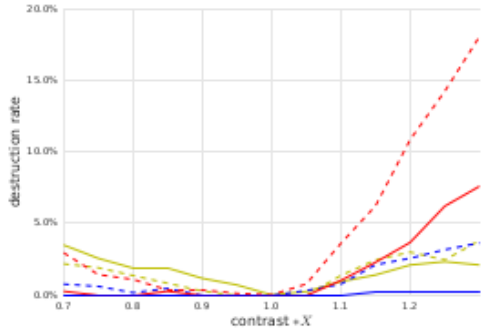
Table 3: Adversarial image destruction rate with photos.

Adversarial method	Average case		Prefiltered case	
	top-1	top-5	top-1	top-5
fast $\epsilon = 16$	12.5%	40.0%	5.1%	39.4%
fast $\epsilon = 8$	33.3%	40.0%	14.6%	70.8%
fast $\epsilon = 4$	46.7%	65.9%	32.4%	91.2%
fast $\epsilon = 2$	61.1%	63.2%	49.5%	91.9%
iter. basic $\epsilon = 16$	40.4%	69.4%	60.0%	87.8%
iter. basic $\epsilon = 8$	52.1%	90.5%	64.7%	91.2%
iter. basic $\epsilon = 4$	52.4%	82.6%	77.5%	94.1%
iter. basic $\epsilon = 2$	71.7%	81.5%	80.8%	96.9%
l.l. class $\epsilon = 16$	72.2%	85.1%	87.5%	97.9%
l.l. class $\epsilon = 8$	86.3%	94.6%	88.9%	97.0%
l.l. class $\epsilon = 4$	90.3%	93.9%	91.9%	98.0%
l.l. class $\epsilon = 2$	82.1%	93.9%	93.1%	98.0%

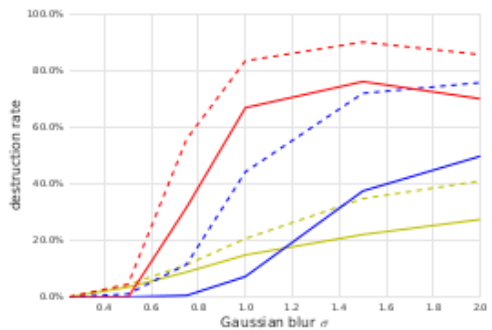
What about other transformations?



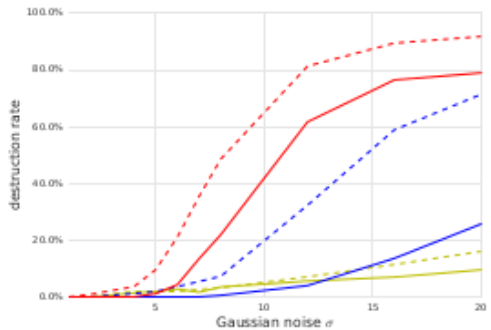
(a) Change of brightness



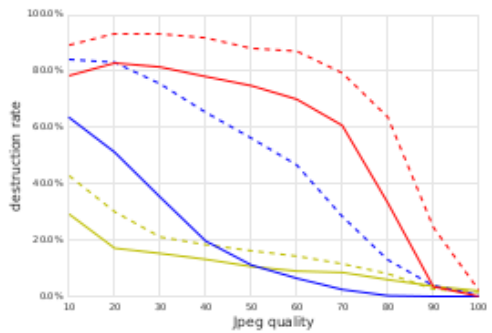
(b) Change of contrast



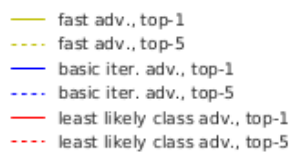
(c) Gaussian blur



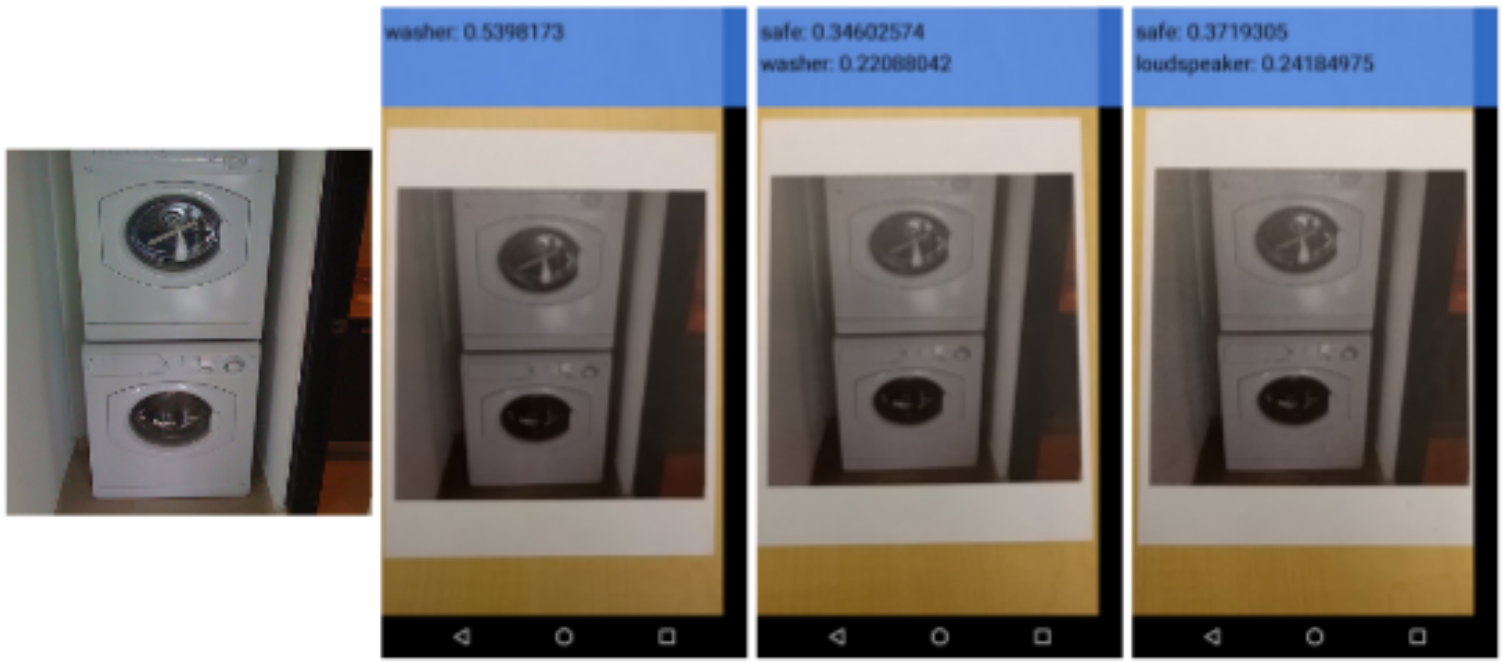
(d) Gaussian noise



(e) JPEG encoding



Adversarial examples transfer



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

Adversarial exmaples constructed for pre-trained ImageNet Inception classifier (Szegedy et. al, 2015) fool the TensorFlow camera demo.

More

- arXiv 1607.02533 https://arxiv.org/abs/1607.02533
- https://www.youtube.com/watch?v=RyEhb-KquEY
- https://www.youtube.com/watch?v=zQ_uMenoBCk