

Stroke-based Character Recognition with Deep Reinforcement Learning

Zhewei Huang^{1,2}, Wen Heng¹, Yuanzheng Tao^{1,2}, Shuchang Zhou¹

¹Megvii Inc(Face++) ²Peking University

hzwer@pku.edu.cn, hengwen@megvii.com, memphis@pku.edu.cn, zsc@megvii.com

Abstract

The stroke sequence of characters is significant for the character recognition task. In this paper, we propose a stroke-based character recognition (SCR) method. We train a stroke inference module under deep reinforcement learning (DRL) framework. This module extracts the sequence of strokes from characters, which can be integrated with character recognizers to improve their robustness to noise. Our experiments show that the module can handle complicated noise and reconstruct the characters. Meanwhile, it can also help achieve great ability in defending adversarial attacks of character recognizers.

1. Introduction

Character recognition is the task that converts the printed or hand-written text into machine-encoded text. This task is usually converted into a multi-category classification task, i.e. one character corresponds to one category.

Currently, most character recognition methods directly predict the labels based on the character images, and no more structure information about characters is considered. Intrinsically, a character is constructed by a sequence of strokes in a certain order. This property can be used to design more robust character recognition method.

Character recognition of natural images is still a challenge to machines. This is because it's really hard for machines to extract efficient information while the characters are with a noisy or bewildering background. It's also shown that the recognition of hand-written digits is easily fooled by adding some noise to the character images, which is called the adversarial attack. One possible way of increasing the robustness to noise is extending the training dataset with these noisy character images to further train the character recognizer. But this approach is costly and it's hard to expose all possible noises to recognizers in training. To alleviate the effects of noise, we can extract the strokes first and reconstruct the characters, then further conduct the recognition.

In this paper, we propose a module to infer stroke from character images, which can be integrated with character recognizers. The inferred stroke sequence can be used in two ways: (1) reconstructing a clean character image for static



Figure 1: Inferred strokes for characters on MNIST (Strokes are dyed for better visualization)

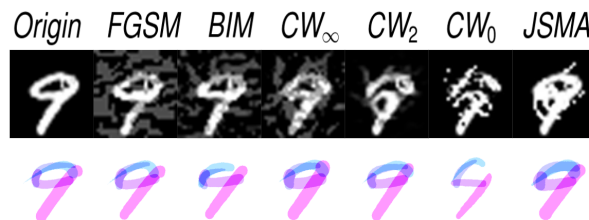


Figure 2: Inferred strokes for adversarial examples of MNIST

image-based character recognizer; (2) taking as the input for stroke sequence based character recognizer. The pipeline of our SCR method is illustrated in Fig. 3.

We regard the stroke inference module as an agent, which can deconstruct a character into a sequence of strokes. We build an efficient agent training environment under the framework of DRL and train an agent. The weighted quadratic Bezier curve is used to simulate the stroke, which is controlled by three weighted points. The agent will learn to predict three weight points to generate a stroke each time, and further, reconstruct a character in sequence. Due to the continuous action spaces in our formulated task, we take the deep deterministic policy gradient algorithm (DDPG) (Lillicrap et al. 2015) to learn the agent. The agent is trained in the manner of weakly supervised learning. We just take the L_2 distance between the reconstructed character and the ground truth as the only supervised loss and don't expose the strokes sequence of each character to the agent. Despite all this, we

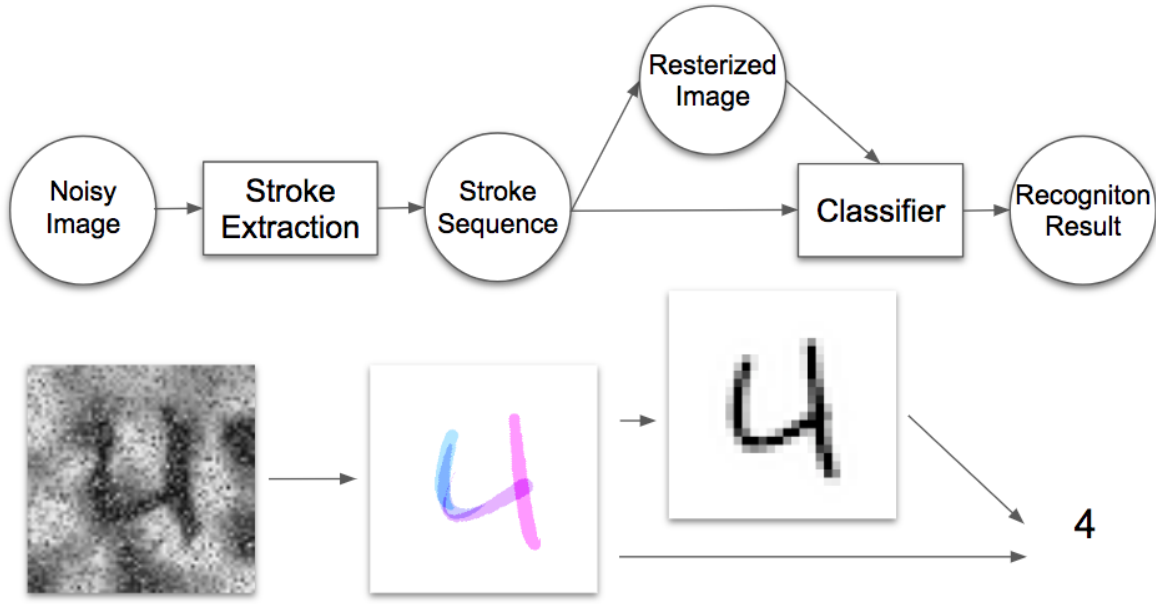


Figure 3: Pipeline of our SCR method

find the agent still can automatically learn to deconstruct a character into a sequence of strokes. And in experiments, we show that the agent can cope with both the Arabic numerals and English letters well.

Our contributions are as follows:

- We propose the stroke inference module to extract strokes from characters, which can be integrated with character recognizer to improve the robustness to noise.
- The stroke inference module is trained under the framework of DRL, with a high-efficiency DDPG.
- Through the experiments on MNIST and SVHN datasets, we show our SCR method have a excellent ability in defending adversarial attacks. Meanwhile, it's also illustrated that the stroke inference module can handle very complicated noise and perfectly reconstruct the characters.

2.Related Work

DRL

The goal of reinforcement learning (RL) is to learn a policy that maps states to actions to achieve the biggest reward. Since the agent trained by deep Q network algorithm (DQN) (Mnih et al. 2015) that combined Q-learning and convolutional neural networks achieved amazing performance from raw pixels in Atari games, many extensions and other DRL algorithms have been proposed to apply DRL to more tasks successfully, such as controlling continuous systems in playing Go (Silver et al. 2016), playing first-person shooter game (Kempka et al. 2016), controlling complex physiologically-based model (Kidziński et al. 2018).

But there have been some papers pointing out that the current DRL research still faces lots of difficulties so reproducing results for state-of-the-art DRL algorithms is seldom straightforward (Henderson et al. 2017). The reward function

is difficult to design and the system is complex so there are many hyper-parameters that will affect the final performance of the agent. Farther more, the difficulty of exploration of environment and the large cost of generating training data makes it hard to apply an algorithm quickly in practice.

Image Reconstruction

Giving agent a way how human constructs images, such as using strokes or simple geometries and train agent to reconstruct images. This topic has always been very hot.

There are some works that focus on the methods of RL that make agent have the ability to infer strokes from images. The work of 2013 studied the automatic real-time generation of simple strokes (Xie, Hachiya, and Sugiyama 2013). DeepMind proposed an approach to train agent with reinforced adversarial learning to master synthesizing programs for images and use strokes to reconstruct images (Ganin et al. 2018).

Adversarial Attack and Defense

Since machine intelligence has been used in security-sensitive applications, robustness started to be important to guarantee the security of application of machine intelligence. Unfortunately, many works have confirmed that deep neural networks are very sensitive to small perturbations of the input vector (Goodfellow, Shlens, and Szegedy 2014; Kurakin, Goodfellow, and Bengio 2016; Carlini and Wagner 2017). Current defense method against adversarial examples follows about three approaches: (1) Training to distinguish normal and adversarial examples. (2) Training with adversarial examples to improve robustness. (3) Preprocessing input data or some other methods to make it difficult to attack target classifier (Meng and Chen 2017). Both (1) and (2) require many adversarial examples to train the models, but

adversarial examples could be generated by unpredictable methods.

3. Learn to Infer Strokes

Problem Definition

Given a distorted character image I_{dis} , our goal is to reconstruct a clean character image I_{rec} that is close to the ground truth character I_{gt} . We are not intending to crop the character from I_{dis} at a time, because this is quite hard for machines, especially when I_{dis} is with severely distortions. Instead, we deconstruct the character into strokes and draw strokes on the canvas in a sequential manner. The procedure is implemented by an agent, which can be learned under the DRL framework.



Figure 4: Simulated strokes using WQBC

We use the **weighted quadratic Bezier curve** (WQBC) to simulate a stroke. The WQBC is determined by three weighted points P_0 , P_1 and P_2 , either of which is denoted as (x, y, w) , where x and y denote the coordinate and w denotes the radius. Then a WQBC is generated as follows:

$$B(t) = (1 - t)^2 P_0 + 2(1 - t)t P_1 + t^2 P_2, 0 \leq t \leq 1$$

We have also tried other curves to simulate strokes. But we find WQBC is simple and flexible enough to simulate the most strokes of common characters.

Agent

We will emphatically introduce how to build an agent under the framework of DRL. The agent will draw a stroke each time on the canvas when given the observation of I_{dis} to reduce the loss between the reconstructed character and I_{gt} . This step goes until the reconstructed character is satisfactory and the agent decides to stop. We will first clarify the terminologies in the formulated task, such as action, state, reward.

Action The action space is the set of actions that the agent can perform. In our formulated task, the action gives three weighted points and a stop signal instruction, which is a 10-dimension real number vector (concatenation of three (x, y, w) , $0 \leq x, y, w \leq 1$ and a binary stop signal indicator). At each step, the agent predicts an action vector. The agent will draw a stroke according to the three weighted points. If the stop signal indicator is positive, then the agent will stop drawing other strokes.

State The state space is constructed by the information that the agent could observe. In our formulated task, the state is formulated as $\{I_{dis}, I_{rec}\}$, where I_{dis} is the distorted character image and I_{rec} is the image from the current canvas. At step 1, I_{rec} is a blank canvas image. At each state, the intensity of I_{dis} and I_{rec} is normalized into the range of $[0, 1]$.

Reward The reward drives the agent to learn to draw strokes to form a character be similar to the ground truth character. We want the similarity to be enhanced after each step, thus a stepwise is designed as follows,

$$r_t = L_t - L_{t+1} \quad (1)$$

where r_t is the reward at step t , L_t and L_{t+1} are the L_2 losses between the canvas and the ground truth image at step t and $t + 1$ respectively. But in practice, we find taking the normalized L_2 loss instead can lead to a better-performed agent. Normalized L_2 loss at step t is calculated by dividing the L_2 loss at step t by the L_2 loss at step 1 (the L_2 loss between the blank canvas and the ground truth character image). Selecting a suitable metric to measure the difference between the canvas and the ground truth image is crucial to learn an efficient agent. Some other metrics can be considered, e.g. SSIM (Wang et al. 2004), Perceptual similarity (Zhang et al. 2018). However, these metrics are mostly designed for natural images and don't work well for the character images.

Training

DDPG Due to the high dimensional and continuous action space, we take DDPG to train the agent. DDPG is easily implemented and with high sampling efficiency. For DDPG, there are two networks: the actor and critic networks. Actor $\pi(s_t)$ gives a policy π that maps a state s_t to action a_t . Critic estimates $Q(s_t, a_t)$ to determine which action a_t is performed when observing state s_t . $Q(s_t, a_t)$ is the accumulated reward until step t .

The critic is trained using Bellman equation and off-policy data:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi(s_{t+1}))$$

where γ is a reward discount factor and $r(s_t, a_t)$ is a reward given by environment when performing action a_t at state s_t .

The actor is trained to maximize the critic's estimated Q .

In other words, actor decides an action for each state and critic predicts an expected reward based on the current canvas and target image for the stroke given by the actor. The critic is optimized to estimate more accurate expected rewards and actor is optimized to maximize the expected rewards.

Architecture We observe that for the actor and critic networks, they all have a part for extracting features from raw pixels. At the same time, the gradient of the actor is very unstable, so it is difficult to converge on a complex network structure. So we share the parameters of low layers of the actor and critic networks, which is abstracted as the visual network. We find that this will significantly speed up the training of the actor and allow us to train more complex network structures.

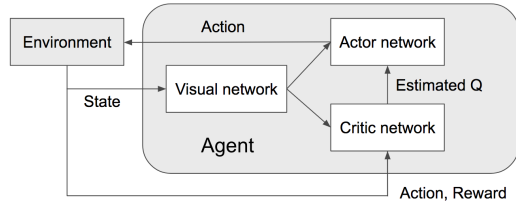


Figure 5: Architecture of the agent

We choose activation function leaky rectified linear unit (Leaky ReLU) (Maas, Hannun, and Ng 2013) for all the hidden layers and perform initialization known as He initialization (He et al. 2015).

Normalization Batch normalization (Ioffe and Szegedy 2015) is a popular technique for accelerating network training, but we found it not very applicable in DRL scenarios. We use weight normalization (Salimans and Kingma 2016) for all layers to make a significant increase in model capabilities. Layer normalization (Ba, Kiros, and Hinton 2016) can also play a similar effect.

Exploration We use parameter space noise (Plappert et al. 2017) ($\sigma = 0.1$) in the parameters of actor to produce actions with structure noise.

Hyper-parameters We use Adam (Kingma and Ba 2014) for learning the neural network parameters with a learning rate of 10^{-4} and 3×10^{-4} for the actor and critic respectively. As in original paper, we use the experience replay memory ($size = 131072$) and the target network ($\tau = 0.001$) for stabilize training and more efficiently use samples (each sample includes state, action, and reward) from the environment.

Implementation Details Our training process runs on a single computer with a GPU. During the training process, the computer spends about half of the time training, and half of the time interacts with the simulators to generate data. When the network interacts with the simulators, each time it generates a mini-batch of actions, the simulators execute them concurrently.

The strokes will be actually simulated on the canvas with a resolution of 256×256 . But when taken as input for networks, we resize all the images into 64×64 pixels. The network structure is shown in the appendix.

The implementation of our method is based on the PyTorch v0.4 library (Paszke et al. 2017). Our simulator can perform 600 strokes and calculate the rewards per second and we can train 40 batches per second. We need about 20 hours to get the converged model each time. In total, we draw 4×10^6 images for each training, which is about 1.5×10^6 batches.

4. Experiment

Datasets and Noisy Characters Generation

MNIST MNIST (LeCun 1998) contains 70,000 examples of hand-written digits, of which 60,000 are training data and

10,000 are testing data. Each example is a grayscale image with a resolution of 28×28 pixels.

SVHN SVHN (Netzer et al. 2011) is a real-world image dataset including over 600,000 digit images. The ‘‘Cropped Digits’’ set is similar to MNIST and each sample is a color image with a resolution of 32×32 pixels. We convert them into grayscale for training and testing.

Synthetic English Letters We choose 4 English letters fonts to generate a dataset containing 104 (4×26) images. Each image shows a printed lowercase English letter and with a resolution of 64×64 pixels.

Noisy Characters Generation The character images in MNIST and the synthetic English letters dataset are taken as the ground truth. We add distortions on the ground truth to generate distorted character images. We mainly consider following distortions. Each pixel value of the image is normalized into the range of $[0, 1]$ at first.

- **Gaussian Noise** We generate a $n \times n$ Gaussian noise ($\sigma = 0.2$), then resize it into the size of the character image and add it on. n is generated as a random power of 2.
- **Gaussian Blur** The radius is a random integer sampled from $\{1, 2, 3, 4, 5\}$.
- **Pepper & Salt Noise** The Signal-to-noise ratio is set as 0.7.
- **Intensity Transform** All pixel values are divided by a random integer sampled from $\{1, 2, 3, 4, 5\}$, and a random real number sampled from $[-0.5, 0.5]$ is added then. And 50% of all images are picked to be reduced by 1.
- **Random Rotation & Crop** The rotation angle is randomly selected from the range of $[-30^\circ, 30^\circ]$. The width of cropped image is 90% of the original image.

Defense against Adversarial Attacks on MNIST

Many works show that the written-digit recognizers are easily fooled by existing adversarial attack methods.

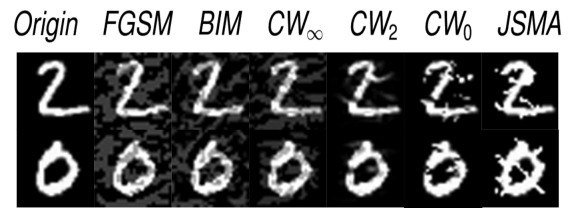


Figure 6: Adversarial examples of MNIST

Based on the MNIST dataset, we make a comparison to an existing defense method, feature squeezing (Xu, Evans, and Qi 2017). As in their paper, we take FGSM (Goodfellow, Shlens, and Szegedy 2014), BIM (Kurakin, Goodfellow, and Bengio 2016), C&W (Carlini and Wagner 2017) and JSMA (Papernot et al. 2016) adversarial attack methods in our experiments and show some adversary examples in Fig 6.

We take the training set (60,000 images) of MNIST to train the agent. These images are taken as the ground truth.

MNIST Experiments	L_∞ Attacks				L_2 Attacks		L_0 Attacks				All Attacks
	FGSM	BIM	CW_∞		CW_2		CW_0		JSMA		
			Next	LL	Next	LL	Next	LL	Next	LL	
No defense	54%	9%	0%	0%	0%	0%	0%	0%	27%	40%	13.00%
Bit Depth (1-bit)	92%	87%	100%	100%	83%	66%	0%	0%	50%	49%	62.70%
Median Smoothing (3x3)	59%	14%	43%	46%	51%	53%	67%	59%	82%	79%	55.30%
SCR (ours)	95%	91%	96%	97%	90%	89%	82%	71%	86%	81%	87.80%

Table 1: Model accuracy against adversarial attack

In training, noisy character images are generated following the noisy character generation procedure.

In Table 1, we show the defense results of our method and feature squeezing, including reducing the color bit depth of each pixel and spatial smoothing. It's shown that our method achieves the best defense results for most adversarial attacks. And the overall performance of our method is significantly better than other methods.

Experiments on synthetic English letters

Training on synthetic English letters is similar to the training on MNIST. We use the same noisy character generation procedure to generate noisy character images in training and testing.

Some stroke inference results are shown in Fig 7.



Figure 7: Results on synthetic English letters

Experiments on SVHN

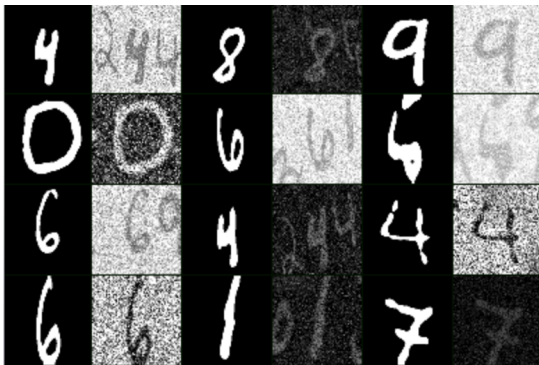


Figure 8: Synthetic SVHN with MNIST as ground truth

The character images of SVHN are cropped from real-scene street view images, and they are with rich noises in nature. Since it's hard for us to obtain the ground truth images (clean and without noise) for SVHN, we take a special strategy to train the stroke inference agent. We synthesize noisy character images based on MNIST as shown in Fig 8. Then take them to train the agent. When testing, we use the trained agent to infer stroke for SVHN. The agent still achieves a higher than 70% recognition rate on SVHN. We believe with explorations on synthesizing more similar images as SVHN, the recognition rate will be further improved. We show some stroke inference results of the character images of SVHN in Fig. 9.

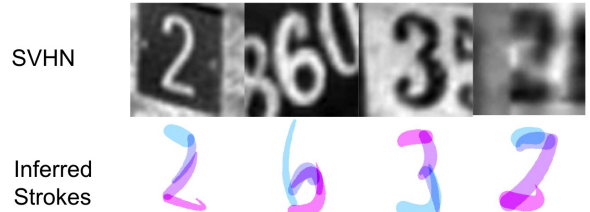


Figure 9: Strokes inference results on SVHN

5. Discussion

We propose a stroke-based SCR method. The core part is the stroke inference module, which is implemented as an agent under the framework of DRL. The experiments show that the stroke inference agent is robust to the noise on characters. And this SCR method shows a great ability in defending adversarial attacks of the hand-written digits.

In our future work, we will pay attention to the following aspects.

- Improve our training algorithm and try some other expressive types of strokes to deal with complex characters such as Chinese, Arabic or calligraphy.
- Extend current method from single-character recognition to multi-character recognition.
- Learn a general agent for multiple characters. It's usually hard to learn an agent for some specific character types, due to the difficulty in obtaining the ground truth characters. We hope to train an agent that can infer strokes for different characters, even without having access to them during training.

References

- [Ba, Kiros, and Hinton 2016] Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.
- [Carlini and Wagner 2017] Carlini, N., and Wagner, D. 2017. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, 39–57. IEEE.
- [Ganin et al. 2018] Ganin, Y.; Kulkarni, T.; Babuschkin, I.; Eslami, S.; and Vinyals, O. 2018. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*.
- [Goodfellow, Shlens, and Szegedy 2014] Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.
- [He et al. 2015] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 1026–1034.
- [Henderson et al. 2017] Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2017. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*.
- [Ioffe and Szegedy 2015] Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.
- [Kempka et al. 2016] Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; and Jaśkowski, W. 2016. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, 1–8. IEEE.
- [Kidziński et al. 2018] Kidziński, Ł.; Mohanty, S. P.; Ong, C.; Huang, Z.; Zhou, S.; Pechenko, A.; Stelmaszczyk, A.; Jarosik, P.; Pavlov, M.; Kolesnikov, S.; et al. 2018. Learning to run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments. *arXiv preprint arXiv:1804.00361*.
- [Kingma and Ba 2014] Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [Kurakin, Goodfellow, and Bengio 2016] Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.
- [LeCun 1998] LeCun, Y. 1998. The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>.
- [Lillicrap et al. 2015] Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
- [Maas, Hannun, and Ng 2013] Maas, A. L.; Hannun, A. Y.; and Ng, A. Y. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, 3.
- [Meng and Chen 2017] Meng, D., and Chen, H. 2017. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 135–147. ACM.
- [Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.
- [Netzer et al. 2011] Netzer, Y.; Wang, T.; Coates, A.; Bischoff, A.; Wu, B.; and Ng, A. Y. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, 5.
- [Papernot et al. 2016] Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z. B.; and Swami, A. 2016. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, 372–387. IEEE.
- [Paszke et al. 2017] Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *NIPS-W*.
- [Plappert et al. 2017] Plappert, M.; Houthoofd, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2017. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.
- [Salimans and Kingma 2016] Salimans, T., and Kingma, D. P. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, 901–909.
- [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. 2016. Mastering the game of go with deep neural networks and tree search. *nature* 529(7587):484–489.
- [Wang et al. 2004] Wang, Z.; Bovik, A. C.; Sheikh, H. R.; and Simoncelli, E. P. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13(4):600–612.
- [Xie, Hachiya, and Sugiyama 2013] Xie, N.; Hachiya, H.; and Sugiyama, M. 2013. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. *IEICE TRANSACTIONS on Information and Systems* 96(5):1134–1144.
- [Xu, Evans, and Qi 2017] Xu, W.; Evans, D.; and Qi, Y. 2017. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*.
- [Zhang et al. 2018] Zhang, R.; Isola, P.; Efros, A. A.; Shechtman, E.; and Wang, O. 2018. The unreasonable effectiveness of deep features as a perceptual metric. *arXiv preprint arXiv:1801.03924*.

Appendix

Network structure

The network structure diagram is shown in Figure 10, Figure 11 and Figure 12, where FC refers to a fully-connected layer, Conv is a convolution layer and Pooling is a 2×2

max pooling. Only the first convolution kernel is 7×7 , the rest are 3×3 . The numbers of convolution filter are 16, 32, 32, 64, 128, 128 respectively. All Leaky ReLU activations between the layers have been omitted for brevity. The activation function after actor and critic network is a linear function. The output of the actor network are clipped into range $[0,1]$.

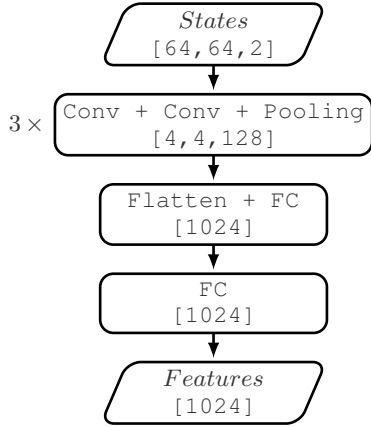


Figure 10: The architecture of the **visual network**.

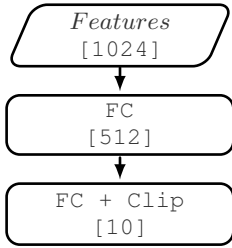


Figure 11: The architecture of the **actor network**.

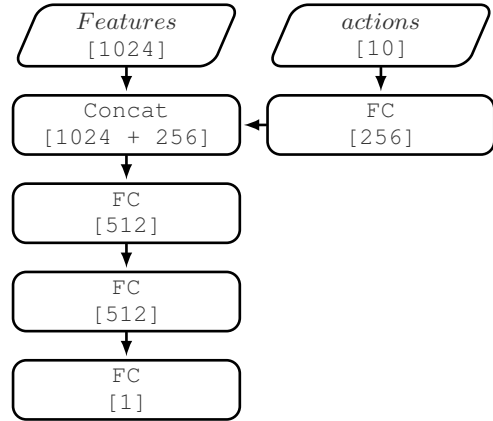


Figure 12: The architecture of the **critic network**.