GAUSSIAN NOISE PASSING CONVOLUTIONAL NEURAL NETWORKS

Shuhao Chang

Department of Electronic Engineering
Tsinghua University
Haidian District, Beijing, China

e-mail: `chang-sh15@mails.tsinghua.edu.cn`

Abstract. Recently, studies on deep neural networks have developed by leaps and bounds, especially convolutional neural networks. When applying deep learning methods to many areas, researchers are often annoyed by the quality of input signal. *Adversarial examples* can largely affect the performance of CNN models. In this paper, the most common and typical interface — additive Gaussian noise is supposed to pass through CNNs and the performance of it is studied in detail, both theoretically and practically. We take simulations and experiments on CIFAR-10 data-set. The experiments show that ReLU and LeakyReLU layers have better training performance than tanh layer when input signal is influenced by Gaussian noise. Besides, max-pooling layer is more stable than average-pooling layer under Gaussian noise.

Keywords: CNN, Gaussian noise, stochastic processes

1 INTRODUCTION

In recent years, deep learning methods specifically represented by convolutional neural networks (CNN) are well studied and widely used in machine learning, intelligent inference as well as pattern recognition areas, which have a profound impact on the development of artificial intelligence. Numbers of network models spring up in this wave[1, 2, 3, 4], gradually excavating an outstanding performance of CNN applying to computer vision and other signal processing areas.

However, the signals we have collected are not always pure, but interfused with various kinds of noise instead. Besides, previous researches have shown that CNNs are susceptible to factitious noise and perform abnormally by the noise of just few pixels in the image[5]. In this paper, we simplified this circumstance and model them as additive Gaussian noise. And we aim to study the transformation of the characteristics of Gaussian noise when passing through each layer of CNN. [6] has done some basic analysis about robust CNN structures under adversarial noise, which has similar purpose as us. This paper is partly based on the result of [6] and probes much deeper into the behaviour of Gaussian noise through CNN. Based on the difference of linearity property, we can divide layers in convolutional neural networks into two classes. We can take the layers in CNN as systems and use stochastic processes methods to deal with the input Gaussian noise.

This paper is organized as follows: Section 2 and 3 respectively studies the change on characteristics of Gaussian noise passing through linear and nonlinear layers of CNN theoretically. Section 4 presents experimental performance and numerical results. The paper concludes in Section 5.

2 GAUSSIAN NOISE THROUGH LINEAR LAYERS

When Gaussian processes pass through linear system, the output will still satisfy Gaussian distribution. We can use this property to derive the output distribution of the system concisely. Two kinds of linear layers are discussed in this subsection.

2.1 Gaussian noise model

We first add randomness into input signals, taking image signals as an example in this paper, and each original pixel can be assumed as a random varible X in $\mathbb{R}^3$ (R, G, B channels). Because of the linear invariance of Gaussian distribution, these new pixels as random varibles still satisfy a shifted normal distribution. We assume the mean value of additive Gaussian noise is 0 and the variance equals a constant matrix Σ. Then the new pixels can be denoted as follows:

$$X_{rgb} \sim N(\mu_{rgb}, \Sigma), \quad X_{rgb}, \mu_{rgb} \in \mathbb{R}^3, \quad \Sigma \in \mathbb{R}^{3\times 3} \tag{1}$$

where μ_{rgb} is the mean vector of the original pixel.

2.2 Convolution layer

Convolution layer is the core of CNNs. It use a certain convolution kernel to match and slide across the pixel matrix and output the sum of multiplication of corresponding positions. Thus, the convolution layer is a kind of linear filter, which can be denoted as follows:

$$Y = \omega^T X + b, \quad X, \omega \in \mathbb{R}^{3ab}, \quad b \in \mathbb{R} \tag{2}$$

where b is the bias and ω is the stretched vector of convolution kernel with height of a pixels and width of b pixels. X is a $a \times b \times 3$ chunk of the input image and Y represents the output value of this layer. Then, we can denote the distribution of Y as:

$$Y \sim N(\omega^T \mu + b, \omega^T \Sigma \omega), \quad \Sigma \in \mathbb{R}^{3ab \times 3ab} \tag{3}$$

We have to emphasize here that the variance Σ here is not equal to that in section 2.1. When the convolution kernel slide across the pixel matrix, the variance of the $3ab$-dimensional chunk will change. In practical application, we often set $a = b$ and use several different convolution kernels to independently filter the image, each exporting only one channel of values.

2.3 Sum-Pooling layer

In essence, the purpose of pooling layer is down-sampling. We aim to obtain more concise and useful information through this layer. When we have clarified the behaviour of Gaussian noise through convolution layer, the situation of sum-pooling layer is much clearer. Because the sum-pooling operation is very similar as convolution layer, only setting the convolution kernel ω as a all-one vector and delete the bias item b. In this case, the first moment of the output will equal to the 1-norm of the original mean value. The distribution of the output Y can be denoted as:

$$Y \sim N(\|\mu\|_1, \|\Sigma\|_1), \quad \Sigma \in \mathbb{R}^{3ab \times 3ab} \tag{4}$$

3 GAUSSIAN NOISE THROUGH NONLINEAR LAYERS

Different with the solution of passing linear layers, it is more complicated to deal with the situation when Gaussian noise passing through non-linear layers. We tend to take advantage of probability distribution function and *order statistics* to obtain the performance of the output. Common nonlinear layers of CNNs are several activate functions and max-pooling function. We will discuss each of them in subsections. Considering the complexity of nonlinear situations, we only calculate the performance of noise through these layers and omit the distribution of input signal. Thus, we have:

$$E(X) = 0, \quad Var(X) = R_X(0) = \sigma^2 \tag{5}$$

3.1 Sigmoid layer

Sigmoid function is a typical function mostly used in machine learning algorithms rather than deep learning situations. For the completeness of this study, however, we still make an analysis here. Sigmoid function can be denoted as follows:

$$y = \frac{1}{1 + \exp(-x)}, \quad x \in (-\infty, +\infty) \tag{6}$$

From the equation, we can infer that sigmoid function on 2-dimension plane is a S shape centrosymmetric curve between $y = 0$ and $y = 1$ with a center at $(0, 1/2)$.

$$\begin{aligned} E(Y) =& E\left(\frac{1}{1+\exp(-x)}\right) = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \frac{\exp\left(-\frac{x^2}{2\sigma^2}\right)}{1+\exp(-x)} dx \\ \approx& \frac{1}{5\sqrt{2\pi}}\left(6 + \frac{1}{\sigma}\right) \end{aligned} \tag{7}$$

$$\begin{aligned} E(Y^2) =& E\left(\frac{1}{1+\exp(-x)}\right)^2 = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \frac{\exp\left(-\frac{x^2}{2\sigma^2}\right)}{(1+\exp(-x))^2} dx \\ \approx& \frac{1}{5\sqrt{2\pi}}\left(6 - \frac{3}{\sigma}\right) \end{aligned} \tag{8}$$

$$Var(Y) = E(Y^2) - E^2(Y) \approx \frac{1}{\pi}\left(-\frac{1}{\sigma^2} - \frac{1}{\sigma} + \frac{4}{5}\right) \tag{9}$$

Here, to simplify the expression, we take a linear approximation of the infinite integral based on the finding that this integral highly fits a straight line when σ varies from 0 to 10.

3.2 Tanh layer

Similar to sigmoid function, *hyperbolic tangent* (tanh) function is still centrosymmetric. However, the center is set at origin point and boundaries are $y = -1$ and $y = 1$.

$$y = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad x \in (-\infty, +\infty) \tag{10}$$

To obtain the gradient expression of tanh function, we have to focus on the inverse function of tanh first.

$$y = \tanh^{-1} x = artanh(x) = \frac{1}{2}\ln\frac{1+x}{1-x}, \quad |x| < 1 \tag{11}$$

Then, we can write down the probability distribution function:

$$\begin{aligned} F_Y(y) =& P(Y(t) \le y) = P(\tanh X(t) \le y) = P(X(t) \le artanh(y)) \\ =& \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{artanh(y)} \exp\left(-\frac{s^2}{2\sigma^2}\right) ds, \quad |y| \le 1 \end{aligned} \tag{12}$$

$$f_Y(y) = \frac{dF_Y(y)}{dy} = \frac{1}{2\sqrt{2\pi}\sigma} \frac{1}{1-y^2} \exp\left(-\frac{artanh^2(y)}{2\sigma^2}\right), \quad |y| \le 1 \tag{13}$$

The mean and variance of the output can be written down as:

$$E(Y) = \int_{-\infty}^{+\infty} \tanh x \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx = 0 \tag{14}$$

$$\begin{aligned} Var(Y) =& \int_{-\infty}^{+\infty} \left(1 - \frac{4}{(e^x+e^{-x})^2}\right) \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \\ =&1 - \int_{-\infty}^{+\infty} \frac{4}{(e^x+e^{-x})^2} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \\ \approx&1 - \frac{\exp(\sigma)-1}{\sqrt{2\pi}\sigma\cosh\sigma} \end{aligned} \tag{15}$$

3.3 ReLU layer

Rectified linear unit (ReLU) layer is the most frequently-used layer in CNNs. ReLU function is a limited linear function which can be denoted as:

$$y = \max(0, x), \quad x \in (-\infty, +\infty) \tag{16}$$

Thus, y equals 0 when x is on the left half side of axis. Actually, in signal processing areas, we call such process as *half-wave linear demodulation*. The probability distribution function of output can be derived as follows:

$$F_Y(y) = P(Y(t) \le y) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{y} \exp\left(-\frac{s^2}{2\sigma^2}\right) ds, \quad y \ge 0 \tag{17}$$

$$f_Y(y) = \frac{dF_Y(y)}{dy} = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2\sigma^2}\right) & , \quad if \quad y \ge 0 \\ 0 & , \quad if \quad y < 0 \end{cases} \tag{18}$$

Thus, the first moment and correlation function are as follows:

$$E(Y) = \int_{0}^{+\infty} x \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx = \sqrt{\frac{1}{2\pi}}\sigma \tag{19}$$

$$\begin{aligned} R_Y(t,s) =& E(Y(t)Y(s)) \\ =& \frac{2\sigma^2(1-\rho^2)^{\frac{2}{3}}}{\pi} \int_0^{\infty} \int_0^{\infty} uv \exp(-(u^2 - 2\rho uv + v^2)) du dv \\ =& \frac{\sigma^2}{2\pi} \left(\sqrt{1-\rho^2} + \rho\left(\frac{\pi}{2} + \arcsin\rho\right)\right) \end{aligned} \tag{20}$$

where $\rho = \frac{R_X(t-s)}{R_X(0)}$. Thus, ReLU layer doesn't change the wide stability of Gaussian noise. The second moment of output is:

$$Var(Y) = E(Y^2) - E^2(Y) = R_Y(0) - E^2(Y) = \left(\frac{1}{2} - \frac{1}{2\pi}\right)\sigma^2 \tag{21}$$

3.4 LeakyReLU layer

To overcome the problem of ReLU layer that neurons fall into the left half side of the axis will lose gradients and *die*, researchers put forward the idea of LeakyReLU layer. Different from ReLU, LeakyReLu function give neurons a little gradient α on the left side. Then the LeakyReLU function can be written as:

$$y = \max(\alpha x, x), \quad x \in (-\infty, +\infty), \quad \alpha > 0 \tag{22}$$

We can use the same thread as section 3.3 to write down the probability distribution function of output first:

$$\begin{aligned} F_Y(y) =& P(Y(t) \le y) \\ =& P(\alpha X(t) \le y)P(X(t) < 0) + P(X(t) \le y)P(X(t) \ge 0) \\ =& \frac{1}{2\sqrt{2\pi}\sigma}\left(\int_{-\infty}^{\frac{y}{\alpha}} \exp\left(-\frac{s^2}{2\sigma^2}\right) ds + \int_{-\infty}^{y} \exp\left(-\frac{s^2}{2\sigma^2}\right) ds\right) \end{aligned} \tag{23}$$

$$f_Y(y) = \frac{dF_Y(y)}{dy} = \frac{1}{2\sqrt{2\pi}\sigma}\left(\frac{1}{\alpha}\exp\left(-\frac{y^2}{2\sigma^2\alpha^2}\right) + \exp\left(-\frac{y^2}{2\sigma^2}\right)\right) \tag{24}$$

The first moment and correlation function are as follows:

$$E(Y) = \left(\alpha\int_{-\infty}^{0} + \int_{0}^{+\infty}\right) x\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right) dx = \sqrt{\frac{1}{2\pi}}\sigma(1-\alpha) \tag{25}$$

$$\begin{aligned} R_Y(t,s) =& E(Y(t)Y(s)) \\ =& \frac{(1-\alpha^2)\sigma^2}{2\pi}\left(\sqrt{1-\rho^2} + \rho\left(\frac{\pi}{2} + \arcsin\rho\right)\right) + \alpha^2\sigma^2\rho \end{aligned} \tag{26}$$

$$Var(Y) = R_Y(0) - E^2(Y) = \left(\frac{1-\alpha^2}{2} - \frac{(1-\alpha)^2}{2\pi}\right)\sigma^2 \tag{27}$$

3.5 ELU layer

Exponential linear units (ELU) is put forward in these years not only to alleviate the vanishing gradient problem via the identity for positive values but also improve the performance of neural networks when noise exists[7]. The ELU function is like follows:

$$y = \begin{cases} x & , \ if \quad x \ge 0 \\ \alpha(\exp(x) - 1) & , \ if \quad x < 0 \end{cases}, \quad \alpha > 0 \tag{28}$$

The probability distribution function is:

$$\begin{aligned} F_Y(y) =& P(Y(t) \le y) \\ =& P(\alpha\exp(X(t) - 1) \le y)P(X(t) < 0) + P(X(t) \le y)P(X(t) \ge 0) \\ =& \frac{1}{2\sqrt{2\pi}\sigma}\left(\int_{-\infty}^{\ln\left(\frac{y}{\alpha}+1\right)} + \int_{-\infty}^{y}\right)\exp\left(-\frac{s^2}{2\sigma^2}\right) ds \end{aligned} \tag{29}$$

$$f_Y(y) = \frac{dF_Y(y)}{dy} = \frac{1}{2\sqrt{2\pi}\sigma}\left(\frac{1}{y+\alpha}\exp\left(-\frac{\ln^2\left(\frac{y}{\alpha}+1\right)}{2\sigma^2}\right) + \exp\left(-\frac{y^2}{2\sigma^2}\right)\right) \tag{30}$$

For the non-linearity of the left half part of ELU function, we can just represent the first and second moment of output concisely.

$$\begin{aligned} E(Y) =& \frac{\alpha}{\sqrt{2\pi}\sigma}\int_{-\infty}^{0}\exp\left(x-\frac{x^2}{2\sigma^2}\right)dx+\sqrt{\frac{1}{2\pi}}\sigma-\frac{\alpha}{2} \\ =& \frac{\alpha}{\sqrt{2\pi}\sigma}\exp\left(\frac{\sigma^2}{2}\right)\int_{-\infty}^{0}\exp\left(-\left(\frac{x}{\sqrt{2}\sigma}-\frac{\sigma}{\sqrt{2}}\right)^2\right)dx+\sqrt{\frac{1}{2\pi}}\sigma-\frac{\alpha}{2} \\ \approx& \frac{\alpha}{2}\left(\exp\left(\frac{\sigma^2}{2}\right)-1\right)+\sqrt{\frac{1}{2\pi}}\sigma \qquad (\sigma \ll \sqrt{2}) \end{aligned} \tag{31}$$

$$\begin{aligned} E(Y^2) =& \frac{1}{\sqrt{2\pi}\sigma}\left(\int_{-\infty}^{0}\alpha^2\left(e^x-1\right)^2+\int_{0}^{+\infty}x^2\right)\exp\left(-\frac{x^2}{2\sigma^2}\right)dx \\ \approx& \alpha^2\left(\exp\left(2\sigma^2\right)-\exp\left(\frac{\sigma^2}{2}\right)+\frac{1}{2}\right)+\frac{\sigma^2}{2} \qquad (\sigma \ll \sqrt{2}) \end{aligned} \tag{32}$$

$$Var(Y) = E(Y^2) - E^2(Y) \approx \alpha^2\exp\left(2\sigma^2\right)+\left(\frac{1}{2}-\frac{1}{2\pi}\right)\sigma^2 \quad (\sigma \ll \sqrt{2}) \tag{33}$$

3.6 Max-Pooling layer

Comparing to sum-pooling layer, max-pooling is more frequently used by researchers when building convolution neural networks. Max pooling simply decreases the dimension of input signal by picking out only the maximum value from part of the matrix. Therefore, the operation can be recognized as taking the infinite norm of the original stretched vector. Then the output can be represented as follows:

$$Y = \|X_{ori}+N\|_\infty = \|X_{in}\|_\infty \tag{34}$$

In this subsection, we still take both original signal and additive Gaussian noise into consideration. Considering the form of max-pooling function, we use *order statistics* to derive the distribution function.

$$\begin{aligned} F_Y(y) =& P(Y(t)\le y) = P(\|X\|_\infty \le y) = P(\max(x_1,x_2,...,x_{3ab})\le y) \\ =& P(x1\le y,...,x_{3ab}\le y) = \prod_{k=1}^{3ab}P(x_k\le y) = (F_x(y))^n \end{aligned} \tag{35}$$

$$f_Y(y) = \frac{dF_Y(y)}{dy} = n(F_x(y))^{n-1}f_x(y) \tag{36}$$

where $3ab$ represents the dimension of pooling kernel. Here, the probability distribution depends on the distribution of input signal, which we cannot write down as a fixed formula. Thus, we do not derive the mean and variance of the output. Additionally, [6] gives a recursion formula of the mean and variance. However, that is much more complex than using *order statistics* methods.

4 EXPERIMENTAL RESULTS

4.1 Gaussian noise through a single layer

In this subsection, we use Matlab to simulate the situation when Gaussian noise passing through a single layer mentioned above each time and see what the distribution of output is. We adopt one image from CIFAR-10 data-set to conduct this simulation. The dimension of images in CIFAR-10 is $32 \times 32 \times 3$. We generate additive Gaussian noise of the same dimension and add it to the original image so as to obtain the simulating input signal, as shown in Fig. 1. The mean and variance of Gaussian noise are 0 and 10 respectively.

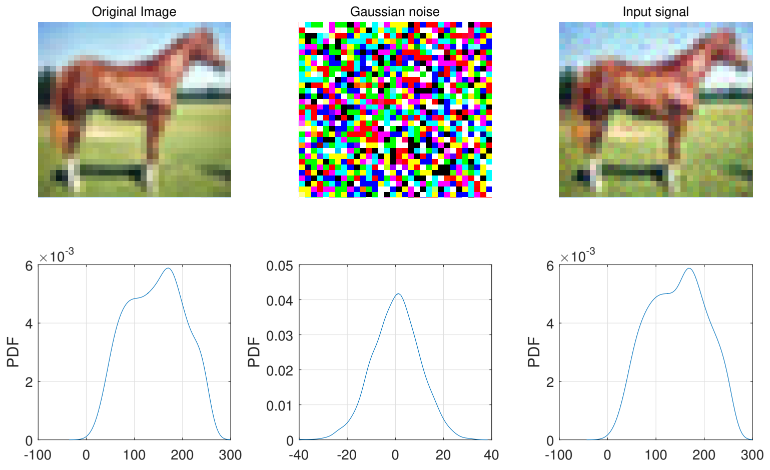

Fig. 1. Adding Gaussian noise to original image

From Fig. 2, we notice that passing through linear layers will not severely change the distribution of input signal. However, the sum-pooling layer uses a all-one kernel to down sampling the input signal, which leads to the phenomenon that distribution of the output of this layer is much more similar then that of convolution layer.

Activate functions and max-pooling function are discussed in Fig. 3. Firstly, we notice that for sigmoid layer and tanh layer the largest density converges on the asymptotic limit, which can be understood easily. For ReLU, LeakyReLU and ELU layers, the outputs perform similarly that most of the probability density converges on the right half side of the axis. We emphasizes here that there does exist negative values of the output from ELU layer. However, the values are very near to 0 because we adopt a little α as the parameter. For the sixth figure, the pdf curve is much similar to that of the input signal, because we use the image signal as input instead of Gaussian noise.

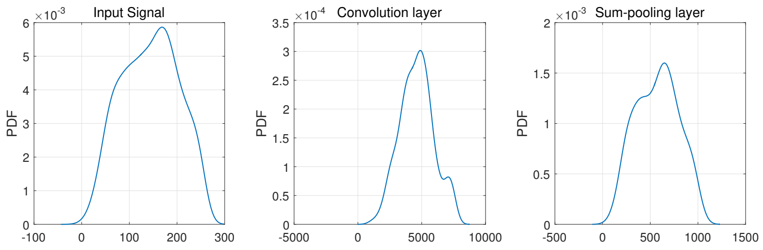

Fig. 2. Gaussian noise passing linear layers

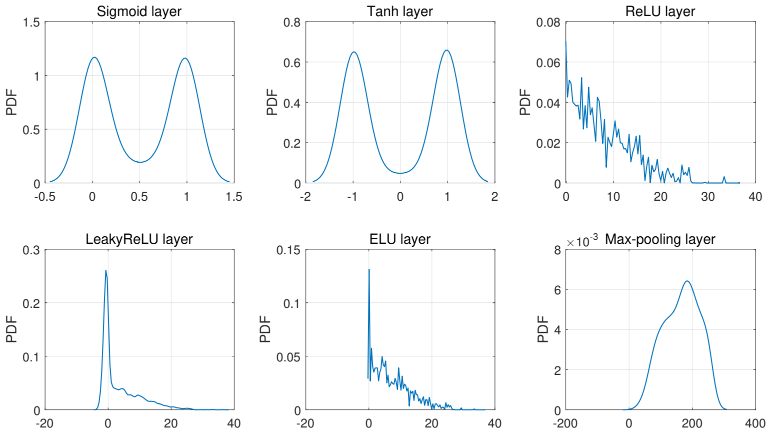

Fig. 3. Gaussian noise passing nonlinear layers

4.2 Gaussian noise through the whole network

In this part, we build a complete convolutional neural network with Python and take practical experiments on it. CIFAR-10 is chosen to be the training and testing data-set. In the experiment, we set other parameters and layer types fixed when taking simulation on a particular combination of layers. We aim to compare the different performances of layers when Gaussian noise passing through them.

The network we build contains five stages, each has two convolution layers, two activate layers and one pooling layer, as shown in Fig. 4. In this experiment, we replace ReLu layer and pooling layer with different activate and pooling functions. Because sigmoid function is more likely to use in machine learning methods instead of deep learning, especially CNNs, we won't compare it with other activate functions here. ELU is a new-born concept and there are not many people tend to use it, which is the reason that we won't add it into this experiment. Here, we only compare the six kinds of combination between three activate functions and two pooling functions.

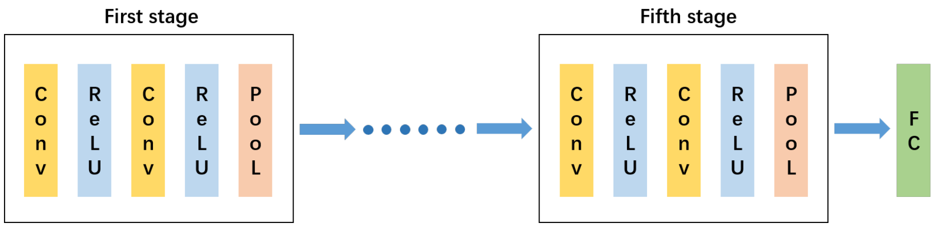

Fig. 4. Network structure in the experiment

Fig. 5 shows the result. First, max-pooling layer has better performance against Gaussian noise than average-pooling layer. As we know, average-pooling is proportional to sum-pooling. So, we use average-pooling layer here to replace max-pooling. The curve shows that when we control the activate functions to be the same, using max-pooling layer always has better test accuracy. Second, tanh layer has better performance in the incipient training but worse performance later. This phenomenon tells us that as the training continues accumulated Gaussian noise has greater impact on the stability of tanh function. Thus, in order to obtain better training effect when input signal is under Gaussian noise, people should use ReLu and LeakyReLU layers more than tanh layers. Third, ReLu and LeakyReLU do not have much difference on performance. Actually, the little gradient on the left half side of LeakyReLU function doesn't influence the stability of training. It just change the convergence time of each training epoch.

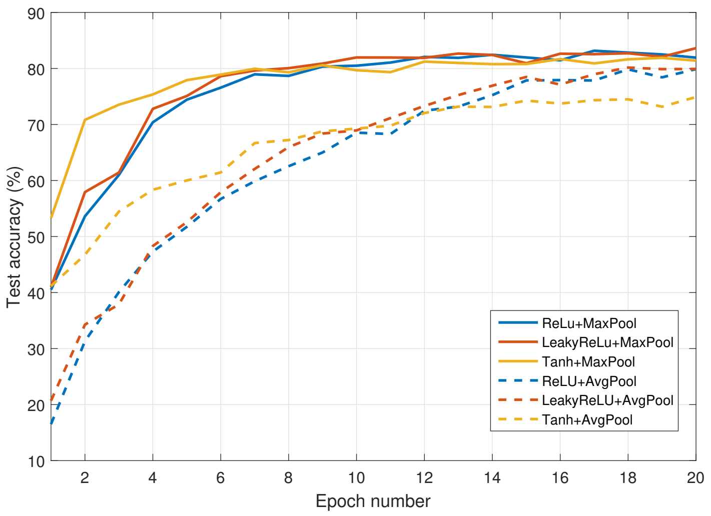

Fig. 5. Gaussian noise passing through different-structured CNNs

5 CONCLUSION

In this paper, we focus on the performance of Gaussian noise when passing through convolutional neural networks and take both theoretical and experimental analysis on it. We not only derive the output distribution of most of the common layers inspired by input signal with additive Gaussian noise, but also prove them conclusively with simulations. The experiments show that ReLu and LeakyReLu layers have better training performance than tanh layer when input signal is influenced by Gaussian noise. Besides, max-pooling layer is more stable than average-pooling layer under Gaussian noise.

REFERENCES

[1] Krizhevsky A, Sutskever I, Hinton G E: *ImageNet classification with deep convolutional neural networks.* International Conference on Neural Information Processing Systems. Curran Associates Inc. 2012:1097-1105.

[2] Szegedy C, Liu W, Jia Y, et al: *Going deeper with convolutions.* Computer Vision and Pattern Recognition. IEEE, 2015:1-9.

[3] Simonyan K, Zisserman A: *Very Deep Convolutional Networks for Large-Scale Image Recognition.* Computer Science, 2014.

[4] He K, Zhang X, Ren S, et al: *Deep Residual Learning for Image Recognition.* 2015:770-778.

[5] Nguyen A, Yosinski J, Clune J.: *Deep neural networks are easily fooled: High confidence predictions for unrecognizable images.* Computer Vision and Pattern Recognition. IEEE, 2015:427-436.

[6] Jin J, Dundar A, Culurciello E.: *Robust Convolutional Neural Networks under Adversarial Noise.* Computer Science, 2015.

[7] Djork-Arn Clevert, Thomas Unterthiner, Sepp Hochreiter, et al.: *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).* ICLR, 2016.

[8] Lee J, Bahri Y, Novak R, et al.: *Deep Neural Networks as Gaussian Processes.* 2017.