

A PROOF: OPTIMISER IS CONTROLLER

A.1 SGD IS A P CONTROLLER

The parameter update rule of SGD from iteration t to $t + 1$ is determined by

$$\theta_{t+1} = \theta_t - r \partial L_t / \partial \theta_t \quad (21)$$

where r is the learning rate. We now regard the gradient $\partial L_t / \partial \theta_t$ as error $e(t)$ in the PID control system Wang et al. [2020]. Compared to the PID controller, we find that SGD can be viewed as one type of P controller with $K_p = r$. The system function of SGD becomes:

$$\theta_{SGD}(s) = r \quad (22)$$

A.2 SGDM IS A PI CONTROLLER

SGDM, which leverages historical gradients, trains a DNN more swiftly than SGD does. The rule of SGDM updating parameter is given by

$$\begin{cases} V_{t+1} = \alpha V_t - r \partial L_t / \partial \theta_t \\ \theta_{t+1} = \theta_t + V_{t+1} \end{cases} \quad (23)$$

where V_t is a term that accumulates historical gradients. $\alpha \in (0, 1)$ is the factor that balances the past and current gradients. It is usually set to 0.9. Dividing two sides of the Equation 23 by α^{t+1} , we get:

$$\frac{V_{t+1}}{\alpha^{t+1}} = \frac{V_t}{\alpha^t} - r \frac{\partial L_t / \partial \theta_t}{\alpha^{t+1}}. \quad (24)$$

Finally, we get θ_{t+1} as follow by iteration:

$$\theta_{t+1} - \theta_t = -r \frac{\partial L_t}{\partial \theta_t} - r \sum_{i=0}^{t-1} \alpha^{t-i} \frac{\partial L_i}{\partial \theta_i} \quad (25)$$

SGDM actually is a PI controller with $K_p = r$ and $K_i = r\alpha^{t-i}$. The system function of SGDM should be:

$$\theta_{SGDM}(s) = r + \frac{r}{s} \cdot \frac{1}{s - \ln(\alpha)} \quad (26)$$

A.3 PID OPTIMISER IS A PID CONTROLLER

SGD and SGDM can be respectively viewed as P and PI controller Wang et al. [2020]. Given that training is often conducted in a mini-batch manner, the learning process is very easy to introduce noise when computing gradients. The proposed PID optimiser Wang et al. [2020] updates network parameter θ in iteration $(t + 1)$ by

$$\begin{cases} V_{t+1} = \alpha V_t - r \partial L_t / \partial \theta_t \\ D_{t+1} = \alpha D_t + (1 - \alpha) (\partial L_t / \partial \theta_t - \partial L_{t-1} / \partial \theta_{t-1}) \\ \theta_{t+1} = \theta_t + V_{t+1} + K_d D_{t+1}. \end{cases} \quad (27)$$

Thus, the θ_{t+1} using PID optimiser is described as follow by iteration:

$$\theta_{t+1} - \theta_t = -r \frac{\partial L_t}{\partial \theta_t} - r \sum_{i=0}^{t-1} \alpha^{t-i} \frac{\partial L_i}{\partial \theta_i} - r K_d \left(\frac{\partial L_t}{\partial \theta_t} - \frac{\partial L_{t-1}}{\partial \theta_{t-1}} \right) \quad (28)$$

where $K_d \left(\frac{\partial L_t}{\partial \theta_t} - \frac{\partial L_{t-1}}{\partial \theta_{t-1}} \right)$ is the D component of the PID controller. The system function of PID should be:

$$\theta_{PID}(s) = r + \frac{r}{s} \cdot \frac{1}{s - \ln(\alpha)} + K_d s \quad (29)$$

When setting the hyperparameter $\alpha = 1.0$, we can get the vanilla PID optimiser: $K_p = r$, $K_i = r$ and $K_d = r \cdot K_d$

A.4 ADAM IS A PI CONTROLLER WITH AN ADAPTIVE FILTER

Based on adaptive estimates of lower-order moments, AdaM algorithm adaptively adjusts the stochastic gradients, and it can be summarized as below:

$$\begin{cases} m_{t+1} = \beta_1 m_t + (1 - \beta_1) \partial L_t / \partial \theta_t \\ v_{t+1} = \beta_2 v_t + (1 - \beta_2) \partial L_t / \partial \theta_t \\ \hat{m}_{t+1} = m_t / (1 - \beta_1^t) \\ \hat{v}_{t+1} = v_t / (1 - \beta_2^t) \\ \theta_{t+1} = \theta_t + \alpha \hat{m}_t / (\sqrt{\hat{v}_{t+1}} + \epsilon) \end{cases} \quad (30)$$

where m_t is the first moment estimate at timestep t , and v_t is the second raw moment estimate. The default set of learning rate α , hyperparameters β_1 , β_2 and ϵ are respectively 0.001, 0.9, 0.999 and 10^{-8} .

The iteration of θ_{t+1} using the AdaM optimizer is described as follows:

$$\begin{aligned} \theta_{t+1} &= \theta_t - r \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \\ &= \theta_t - r \cdot \frac{\frac{\sum_{i=0}^t \beta_1^{t-i} (\partial L_i / \partial \theta_i)}{\sum_{i=1}^t \beta_1^{i-1}}}{\sqrt{\frac{\sum_{i=1}^t \beta_2^{t-i} (\partial L_i / \partial \theta_i)^2}{\sum_{i=1}^t \beta_2^{i-1}} + \epsilon}} \\ &= \theta_t - r \cdot \frac{1}{M} \beta_1^0 \frac{\partial L_t}{\partial \theta_t} - r \cdot \frac{1}{M} \sum_{i=0}^{t-1} \beta_1^{t-1-i} \frac{\partial L_i}{\partial \theta_i} \end{aligned} \quad (31)$$

where M is the adaptive part of AdaM, and its formula is:

$$M = \frac{1}{\sqrt{\frac{\sum_{i=0}^t \beta_2^{t-i} (\partial L_i / \partial \theta_i)^2}{\sum_{i=0}^t \beta_2^{i-1}} + \epsilon}} \cdot \frac{1}{\sum_{i=0}^t \beta_1^{i-1}} \quad (32)$$

Compared to Equation 25, the adaptive component M of AdaM plays an important role on adapting the learning system. We cannot derive the system function of AdaM, as the high complexity of M . Finally, we directly use the same S function in SIMULINK and get its system response on above mentioned ANN models.

A.5 FILTER PROCESSED SGD

Although Gaussian LPF-SGD outperforms other SGD variants, we still do not know which part it has filtered, for example, high frequency, low frequency or any band frequency parts. In this study, we summarize the SGD learning process under the processing of filters as below:

$$\begin{cases} \widehat{\partial L_t / \partial \theta_t} = \partial L_t / \partial \theta_t + \partial \left(\int_{-\infty}^{\infty} L(\theta_t - \tau) H(\tau) d\tau \right) / \partial \theta_t \\ \theta_{t+1} = \theta_t - r \widehat{\partial L_t / \partial \theta_t} \end{cases} \quad (33)$$

where H is a Gaussian kernel and $L(\theta_t)$ is the loss function of the training process in GLPF-SGD Bisla et al. [2022]. The θ_{t+1} using *Filter* processed SGD optimiser is described as follow by iteration:

$$\begin{aligned}
\theta_{t+1} &= \theta_t - r \frac{\partial L_t}{\partial \theta_t} + r \frac{\partial \left(\int_{-\infty}^{\infty} L(\theta_t - \tau) H(\tau) d\tau \right)}{\partial \theta_t} \\
&= \theta_t - r \frac{\partial L_t}{\partial \theta_t} + r \frac{\partial (L \otimes H)}{\partial \theta_t} \\
&= \theta_t - r \frac{\partial L_t}{\partial \theta_t} + r \frac{1}{G} \frac{\sum_{i=0}^N \partial (L(\theta_t - \tau_i) H(\tau_i))}{\partial \theta_t}
\end{aligned} \tag{34}$$

where G is the gain of the filter H with the order of N , and \otimes is the convolution process. Finally, the system function of filter processed SGD becomes to:

$$\theta_{FP-SGD}(s) = r \left(Gain \cdot \frac{\prod_{i=0}^m (s + h_i)}{\prod_{j=0}^n (s + l_j)} \right) \tag{35}$$

In this study, to analyse which frequency parts are beneficial to the training, we used a second-order Infinite Impulse Response (IIR) filter instead of the Gaussian kernel filter. By approximately setting the cutoff frequency at half, we imply a low-pass filter ranging from 0 Hz to half the sampling rate and a high-pass filter from half the sampling rate up to the sampling rate. Consequently, knowledge of the exact sampling rate is unnecessary, and essentially, it remains unobtainable.

B PROOF: LEARNING SYSTEMS OF MOST ANNS ARE CONTROL SYSTEMS

B.1 CNN AND ITS CONTROL SYSTEM

Most CNNs have been used to perform the classification task using the backpropagation algorithm. Obviously, this learning system is a single-input-single-output (SISO) control system, indicating that each sample corresponds to a single label. Figure 6 provides a concise representation of the learning structure when focusing solely on the optimizer, exemplified here by the fuzzyPID optimizer applied to CNNs. If considering only optimiser, its brief learning structure can be seen in Figure 6, and we give an example of using fuzzyPID optimiser on CNNs. When using each *Controller*, the Laplace transform of the learning process becomes:

$$\begin{aligned}
\theta(s) &= Controller \cdot E(s) \\
&= Controller \cdot \left(\frac{\theta^*}{s} - \theta(s) \right) \\
&= \frac{Controller}{Controller + 1} \cdot \frac{\theta^*}{s}
\end{aligned} \tag{36}$$

Backpropagation algorithm based ANN models rely on the backward error to update weights themselves, and inevitably, the system function of their learning systems have been determined by such designed algorithm. Therefore, there are two factors can significantly affect their performance. One is hyperparameter that setups high techniques on ANN models, and another one is optimiser that controls the convergence speed and stability.

It is clear that the network parameter update using SGD optimiser depends on current gradient $r \partial L_t / \partial \theta_t$, but other well-performed updating methods, such as SGDM, AdaM and PID, have considered the previous gradient. The accumulation part of gradients in SGDM can accelerate the learning process, and the introduction of decay term α is to keep the gradients away from the current value so that it can alleviate noise. Building on SGDM, the PID optimizer introduces the predicted

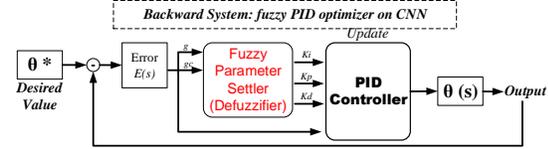


Figure 6: The control system of CNN updated by the FuzzyPID optimiser.

future trend (the difference between the current gradient and the previous one) to adjust the learning process, and its speed becomes faster than SGDM. However, coefficients of PID optimiser, such as P, I and D, are totally fixed, and that will bring another problem – overshooting. To counteract this issue, we used fuzzy logic to adaptively adjust the coefficients of PID optimiser. Inspired by the GLPF-SGD, we believe the learning process using any optimiser relies on specific frequency components. In this study, we designed two filters to figure out which frequency component ANN models prefer. To avoid a long lag of convolution computing, we only applied a second-order IIR filter on the SGD learning process. Even without the exact sampling rate, we have chosen from the half, as the frequency component has no relationship with the sampling rate if we cutoff from the 2^{-1*i} of sampling rate. Therefore, we can get determined $\theta(s)$ of backpropagation based learning systems using various optimisers as follow:

(1) When *Controller* = $\theta_{SGD}(s)$, we can get $\theta(s)$ of backpropagation based CNNs as below:

$$\theta(s) = \frac{K_p}{K_p + 1} \cdot \frac{\theta^*}{s} \quad (37)$$

(2) When *Controller* = $\theta_{SGDM}(s)$, and if we set $\alpha = 1.0$, we can get $\theta(s)$ of backpropagation based CNNs using SGDM as the optimiser as below:

$$\theta(s) = \frac{K_p s + K_i}{(K_p + 1)s + K_i} \cdot \frac{\theta^*}{s} \quad (38)$$

(3) Based on prior knowledge of control system engineering, PID optimiser was proposed by adding D component on SGDM optimiser. According to the analysis of PID optimiser Wang et al. [2020] and the Ziegler–Nichols optimum setting rule Ziegler & Nichols [1942], we also set $P = 1$, $I = 5$ and $D = 100$ here. Therefore, when *Controller* = $\theta_{PID}(s)$, $\theta(s)$ can be computed by:

$$\begin{aligned} \theta(s) &= \frac{K_p + K_i \frac{1}{s} + K_d s}{K_p + K_i \frac{1}{s} + K_d s + 1} \cdot \frac{\theta^*}{s} \\ &= \frac{K_d s^2 + K_p s + K_i}{K_d s^2 + (K_p + 1)s + K_i} \cdot \frac{\theta^*}{s} \end{aligned} \quad (39)$$

(4) Considering the use of fuzzy logic on PID optimiser, we finally get Equation 13 that can compute the system response of FuzzyPID on backpropagation based ANNs.

(5) When using AdaM as the optimiser, apart from the adaptive part, we found AdaM shares the same parameter updating strategy as the SGDM. With $\beta_1 = 0.9$, the Laplace transform of AdaM becomes:

$$AdaM(s) = \frac{1}{M} \cdot K_p + \frac{1}{M} \cdot K_i \frac{1}{s} \cdot \frac{1}{s - \ln(\beta_1)} \quad (40)$$

where the Laplace transform of the last term in Equation 28 becomes to:

$$\begin{aligned} \text{Laplace} \left(\sum_{i=0}^{t-1} \beta_1^{t-1-i} \frac{\partial L_i}{\partial \theta_i} \right) &= \text{Laplace} \left(\sum_{i=0}^{t-1} e^{\ln \beta_1 (t-1-i)} \right) \cdot \text{Laplace} \left(\sum_{i=0}^{t-1} \frac{\partial L_i}{\partial \theta_i} \right) \\ &= \frac{1}{s - \ln(\beta_1)} \cdot \frac{1}{s} \end{aligned} \quad (41)$$

Hence, the system function $\theta(s)$ when using AdaM as the optimiser is:

$$\begin{aligned} \theta(s) &= \frac{AdaM(s)}{AdaM(s) + 1} \cdot \frac{\theta^*}{s} \\ &= \frac{K_p s + K_i}{M s^2 + (K_p - M \ln \beta_1) s + K_i} \cdot \frac{\theta^*}{s} \end{aligned} \quad (42)$$

(5) Additionally, when $Controller = \theta_{FP-SGD}(s)$ the system function of using SGD processed with $Filter$ is defined as:

$$\begin{aligned}\theta(s) &= \frac{Filter}{Filter + 1} \cdot \frac{\theta^*}{s} \\ &= \frac{Gain \cdot \prod_{i=0}^m (s + h_i)}{Gain \cdot \prod_{i=0}^m (s + h_i) + \prod_{j=0}^n (s + l_j)} \cdot \frac{\theta^*}{s}\end{aligned}\quad (43)$$

B.2 FFNN AND ITS CONTROL SYSTEM

FFNN Hinton [2022], based on the forward-forward algorithm mainly aims to visualize the learning process. For a clear analysis on FFNN, we set the portion of positive samples $\lambda = 0.5$ and the threshold $Th = 1.0$. These two hyperparameters were used to make the goodness be well above some threshold value for real data and well below that value for negative data. Essentially, we still use the backpropagation algorithm to update the weights for each layer, as one method used in Hinton [2022]. Based on Equation 14 and 15, we found when $\lambda = 0.5$, the optimal result θ^* has no relationship with the learning system. We analysed FFNN on seven optimisers (e.g., SGD, SGDM, AdaM, PID, LPF-SGD, HPF-SGD and FuzzyPID) and derived their control system functions as below:

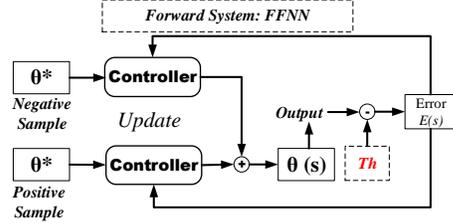


Figure 7: The control system of FFNN.

$$\begin{aligned}\theta(s) &= \left(-(1 - \lambda) \frac{\theta^*}{s} + \lambda \frac{\theta^*}{s} - \left[\theta(s) - \frac{Th}{s} \right] \right) \cdot Controller \\ &= \frac{1}{Controller + 1} \cdot \left(\frac{(2\lambda - 1)\theta^* + Th}{s} \right)\end{aligned}\quad (44)$$

But when $\lambda \neq 0.5$ and the threshold $Th \neq 1.0$, the system function and its classification performance will be influenced by these two hyperparameters. The product of $(2\lambda - 1)\theta^* + Th$ is a gain adjustment part that will affect the learning process.

B.3 GAN AND ITS CONTROL SYSTEM

The essence of GAN is that G and D play games with each other and finally reach a Nash equilibrium point Kreps [1989], but this is only an ideal situation. The normal situation is that it is easy for one party to be strong and the other party to be weak. Therefore, two problems appeared (1) Gradient disappearance and (2) mode collapse corresponding to D and G being the result of the stronger side.

The situation of gradient vanishing is that D wins the game. Because the gradient update of G comes from D, and in the initial stage of training, the input of G is randomly generated noise, which will definitely not generate good pictures, but D performs well. It is easy to judge the true and false samples, that is, there is almost no loss in the training of D. Therefore, there is no effective gradient information back to G itself.

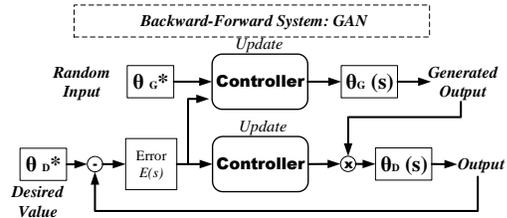


Figure 8: The control system of classical GAN.

The problem of mode collapse is mainly that G wins the game. That causes D to be unable to distinguish between real pictures and fake pictures generated by G. But D cannot tell the difference, and give the correct evaluation, then G will think

that this picture is correct. Thus, D still gives the correct evaluation. Therefore, these two ANNs are such mutual deception.

For a clear analysis on GAN, we used a classical GAN model Goodfellow et al. [2014]. We derive its control system function (Seen from Figure 8) under seven optimisers:

(1) When using SGD as the optimiser, $Controller = K_p$ we get the the control system of G as below:

$$\theta_G(s) = \frac{1}{2} \cdot \left(\frac{\theta_D^*}{K_p} \pm \sqrt{\left(\frac{\theta_D^*}{K_p} \right)^2 - \frac{4}{s}} \right) \quad (45)$$

(2) When using SGDM (a PI controller) as the optimiser, $controller = K_p + K_i/s$, we get the the control system of G as below:

$$\theta_G(s) = \frac{1}{2} \cdot \left(\frac{\theta_D^* s}{K_p s + K_i} \pm \sqrt{\left(\frac{\theta_D^*}{K_p s + K_i} \right)^2 - \frac{4}{s}} \right) \quad (46)$$

(3) When using AdaM (merging the PI and an adaptive filter) as the optimiser, $controller = AdaM(s)$, we get the the control system of G as below:

$$\theta_G(s) = \frac{1}{2} \cdot \left(\frac{\theta_D^*}{AdaM(s)} \pm \sqrt{\left(\frac{\theta_D^*}{AdaM(s)} \right)^2 - \frac{4}{s}} \right) \quad (47)$$

(4) When using PID (considering the pass, current and future) as the optimiser, $controller = K_p + K_i/s + K_d s$. Finally, we get the control system of G as below:

$$\theta_G(s) = \frac{1}{2} \cdot \left(\frac{\theta_D^*}{K_p + \frac{K_i}{s} + K_d s} \pm \sqrt{\left(\frac{\theta_D^*}{K_p + \frac{K_i}{s} + K_d s} \right)^2 - \frac{4}{s}} \right) \quad (48)$$

(5) When using *Filter* processed SGD as the optimiser, $controller = Filter$, we get the the control system of G as below:

$$\theta_G(s) = \frac{1}{2} \cdot \left(\frac{\theta_D^*}{Gain \cdot \frac{\prod_{i=0}^m (s+h_i)}{\prod_{j=0}^n (s+l_j)}} \pm \sqrt{\left(\frac{\theta_D^*}{Gain \cdot \frac{\prod_{i=0}^m (s+h_i)}{\prod_{j=0}^n (s+l_j)}} \right)^2 - \frac{4}{s}} \right) \quad (49)$$

Owing to the complexity of their system functions, we finally decided to use MATLAB SIMULINK to analyse their system response and stability, as shown in Figure 5.

C RESIDUAL CONNECTIONS

Residual connections (RSs) He et al. [2016b]; Eunice et al. [2022] aim to ease the training of DNNs, and it can (1) increase the depth of ANNs and (2) avoid gradient vanishing. Most state-of-the-art (SOTA) ANN models have RSs, but the use of such forward connections has no systematic analysis. Compared to a plain CNN layer, Residual Block adds a short cut from the input features to the output of the mapping. The output from the residual block is $H(x) = f(x) + x$, where input features are x , the output from the original mapping is $f(x)$, then, our desired output is $H(x)$. We are learning the residuals from the output in relate to the input, as the RS is trying to fit the mapping $f(x) = H(x) - x$.

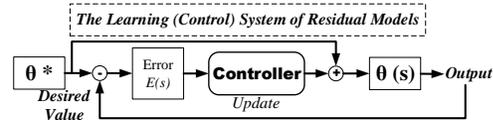


Figure 10: Control system of Residual models.

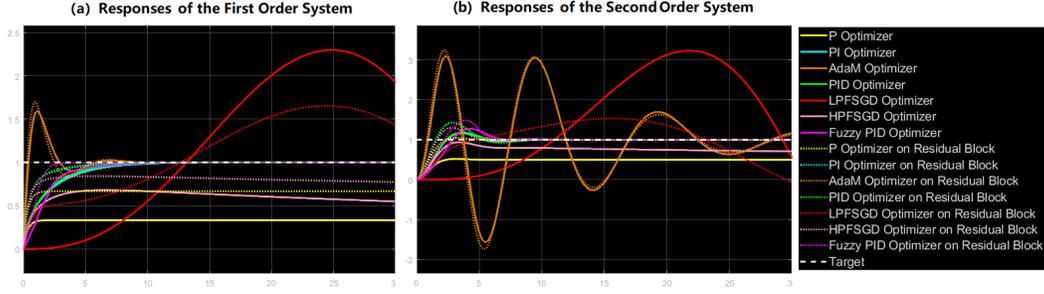


Figure 9: Step responses of models with residual connections across various optimizers: such as SGD, SGDM, AdaM, PID, LPF-SGD, HPF-SGD and FuzzyPID optimisers.

In this study, the models we designed with two or four hidden layers constitute a first-order system, and their system response can also be seen in Figure 9(a). However, we assume SOTA models, such as VGG19 Simonyan & Zisserman [2014], ResNet18, ResNet50, ResNet101 He et al. [2016a], DenseNet121 Zhu & Newsam [2017], MobileNetV2 Sandler et al. [2018], EffecientNet Tan & Le [2019], are second-order (or higher) models. According to the computing process of SGD and the explanation of ResNet He et al. [2016b], we present a reformulated RS mechanism below (ignoring BN Ioffe & Szegedy [2015], ReLU Nair & Hinton [2010], pooling Wu & Gu [2015], and exponential or cosine decay Li et al. [2021]):

$$\frac{\partial L_t}{\partial \hat{\theta}_t} \frac{\partial \hat{\theta}_t}{\partial \theta_t} = \frac{\partial L_t}{\partial \theta_t} \left(1 + \frac{\partial}{\partial \theta_t} \sum_{i=1}^{L-1} \mathcal{F}(\theta_i) \right) \quad (50)$$

where \mathcal{F} is a residual function, $\hat{\theta}_t$ is the weights in the residual block, and L (also interpretable as the depth of a single residual block) is the deeper unit in He et al. [2016b]. Equation 50 indicates that the gradient $\partial L_t / \partial \theta_t$ can be decomposed into two additive terms: a term of $\partial L_t / \partial \theta_t$ that propagates information directly without concerning any weight layers, and another term of $\frac{\partial}{\partial \theta_t} \sum_{i=1}^{L-1} \mathcal{F}(\theta_i)$ that propagates through the weight layers. The additive term of (or this direct component) $\partial L_t / \partial \theta_t$ determines that the learning system will consider information which directly propagates back to θ_t . The parameter update rule of SGD from iteration t to $t + 1$ using RSs is determined by :

$$\theta_{t+1} = \theta_t - r \partial L_t / \partial \theta_t - r \quad (51)$$

where we assume the residual block \mathcal{F} is a simple block. Thus, we finally get the system function of residual connections $\theta_{RS}(s)$ as below:

$$\theta_{RS}(s) = r + \frac{r}{s} \quad (52)$$

where the learning rate r can be served as K_p , and $\frac{r}{s}$ is aligns with the second part $\frac{K_i \alpha^{t-i}}{s}$ of SGDM in Equation 25. The difference is that SGDM has a momentum that takes previous gradients into account, but RS integrates information from preceding layers. Analysing a node within RS-based ANN models, we found the system function of these two – SGDM and RS – have a very similar format. SGDM optimizes the weight of models by accumulating previous gradients with the use of a momentum factor to adjust the effect of accumulation on the time dimension. However, RS optimizes the model by adding passed information to the current block on the space dimension.

In Figure 14, models with residual connections, such as ResNet50, DenseNet121, ModelNetV2, and EffecientNet, have the classification advantage using SGDM and PID, even though PID displays irregularities in the training curve. Interestingly, this observation is also echoed in Figure 9(b), as the rising time of SGDM and PID controller on residual connections is shorter than others (except AdaM, although AdaM can rise very fast, it demonstrates heightened oscillations). FuzzyPID trails

closely, while LPF-SGD lags due to its pronounced low-frequency characteristics leading to the most gradual climb.

D CYCLEGAN

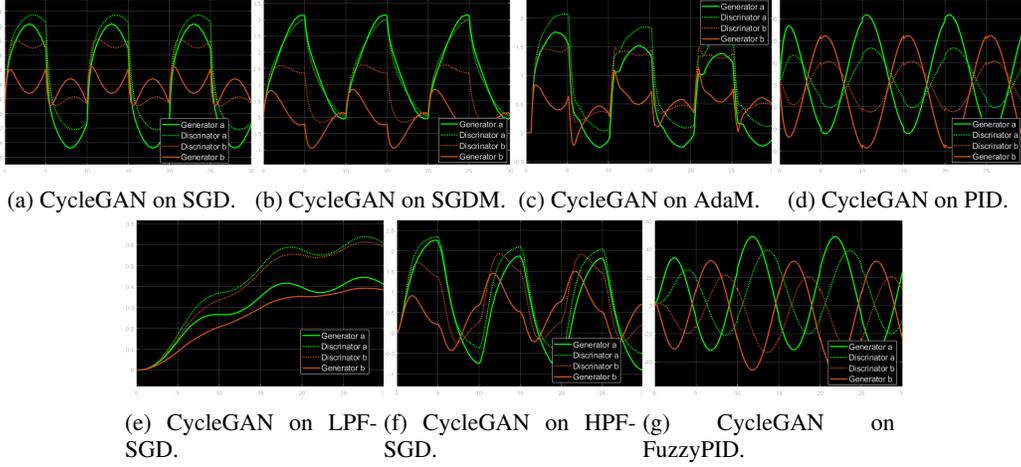


Figure 11: The system response of CycleGAN on different hyperparameters and optimisers, such as SGD, SGDM, AdaM, PID, LPF-SGD, HPF-SGD and FuzzyPID optimisers.

CycleGAN Zhu et al. [2017] aims to translate an image from a source domain a to a target domain b in the absence of paired examples. We denote the data distribution as $a \sim p_{data}(a)$ and $b \sim p_{data}(b)$. CycleGAN contains two mapping functions $G_a: A \rightarrow B$ and $G_b: B \rightarrow A$, and associated adversarial discriminators D_a and D_b . D_b encourages generator G_a to translate A into outputs indistinguishable from domain B , and vice versa for D_A and B . According to its learning system, we present the control system of CycleGAN in Figure 12. CycleGAN has two Generators and two Discriminators, and taking the cycle consistency loss into account, its loss function has three parts as below:

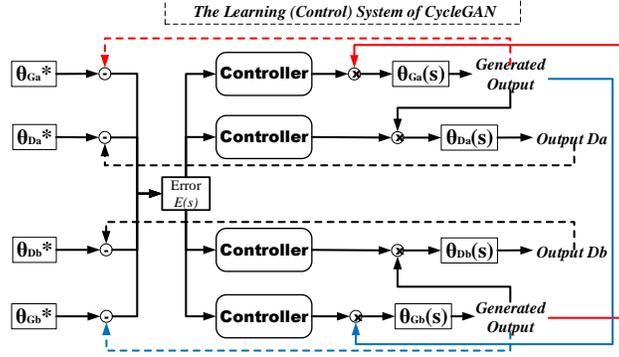


Figure 12: The control system of CycleGAN.

$$\mathcal{L}(G_a, G_b, D_a, D_b) = \mathcal{L}_{GAN}(G_a, D_b, A, B) + \mathcal{L}_{GAN}(G_b, D_a, B, A) + \lambda \mathcal{L}_{cyc}(G_a, G_b) \quad (53)$$

where for the mapping function $G_a: A \rightarrow B$ and its discriminator D_b , we express the objective as:

$$\mathcal{L}_{GAN}(G_a, D_b, A, B) = \mathbb{E}_{a \sim p_{data}(a)}[\log D_b(b)] + \mathbb{E}_{b \sim p_{data}(b)}[\log(1 + D_b(G(a)))] \quad (54)$$

For each image b from domain B , G_a and G_b should satisfy backward cycle consistency: $b \rightarrow G_a(b) \rightarrow G_a(G_b(b)) \approx b$. Thus, the cycle consistency loss should be:

$$\mathcal{L}_{cyc}(G_a, G_b) = \mathbb{E}_{a \sim p_{data}(a)}[\|G_b(G_a(a)) - a\|_1] + \mathbb{E}_{b \sim p_{data}(b)}[\|G_a(G_b(b)) - b\|_1] \quad (55)$$

CycleGAN used the L1 norm in this loss with an adversarial loss between $G_b(G_a(a))$ and a , and between $G_a(G_b(b))$ and b , but did not observe improved performance. Therefore, we get the system function of CycleGAN as below:

$$\theta_{Da}(s) = controller \cdot \theta_{Ga}(s) \cdot E(s) \quad (56)$$

$$\theta_{Ga}(s) = controller \cdot E(s) \quad (57)$$

$$\theta_{Db}(s) = controller \cdot \theta_{Gb}(s) \cdot E(s) \quad (58)$$

$$\theta_{Gb}(s) = controller \cdot E(s) \quad (59)$$

$$E(s) = \left[\frac{\theta_{Da}^*}{s} - \theta_{Da}(s) \right] + \left[\frac{\theta_{Db}^*}{s} - \theta_{Db}(s) \right] + \left[\frac{\theta_{Ga}^*}{s} - \theta_{Ga}(s)\theta_{Gb}(s) \right] + \left[\frac{\theta_{Gb}^*}{s} - \theta_{Gb}(s)\theta_{Ga}(s) \right] \quad (60)$$

We simulated the system response of an advanced GAN – CycleGAN on seven controllers (optimisers) and summarized the result in Figure 11. PID and FuzzyPID controllers can generate the excellent stable sinusoidal signals both on G_a and G_b . SGDM controller failed to generate sinusoidal signals, otherwise, SGD and AdaM can generate acceptable sinusoidal signals. For the generated MNIST in Figure 13, after 100 epochs training, PID can generate 100% correct samples both from G_a to G_b and from G_b to G_a . Notably, the ability of CycleGAN to produce samples from a single dataset was significantly enhanced when utilizing the FuzzyPID, which yielded flawless samples from the out-set. This suggests that FuzzyPID might be the optimal choice for optimizing the learning updates of CycleGAN. The generated samples are depicted in Figure 13. A manual evaluation of the alignment between samples from and vice versa was also conducted. Preliminary observations indicate that the PID and FuzzyPID optimisers outshine the others when applied to models that utilize a cycle consistency loss, such as CycleGAN.

E HYPERPARAMETERS

Table 3: Hyper-parameters for the image classification task on MNIST.

Hyper-parameter	Backward System	Forward System	Backward-Forward System
Data augmentation	Auto	Auto	Auto
Input resolution	[28,28,1]	[28,28,1]	[28,28,1]
Epochs	40	200	200
Batch size	100	100	100
Hidden dropout	0	0	0
Random erasing prob	0	0	0
EMA decay	0	0	0
Cutmix α	0	0	0
Mixup α	0	0	0
Cutmix-Mixup	0	0	0
Label smoothing	0.1	0.1	0.1
Peak learning rate	2e-3	2e-3	2e-4
Steps per block	/	60	/
Positive samples portion λ	/	[0.3, 0.5, 0.7]	/
<i>Threshold</i>	/	[0.1, 1.0, 10.0]	/
optimiser	{SGD, SGDM, AdaM, PID, LPF-SGD, HPF-SGD, FuzzyPID}		

In this study, we conducted three primary experiments, as detailed in Table 3. We roughly separated them to experiments on Backward System, Forward System, Backward-Forward System. Experiments on Backward System and Backward-Forward System do not have Steps per block, *Threshold* and Positive samples portion. However, to make a fair comparison, all experiment should use seven optimisers on the same hyperparameters. These experiments were categorised based on the Backward System, Forward System, and Backward-Forward System. Notably, the Backward System and

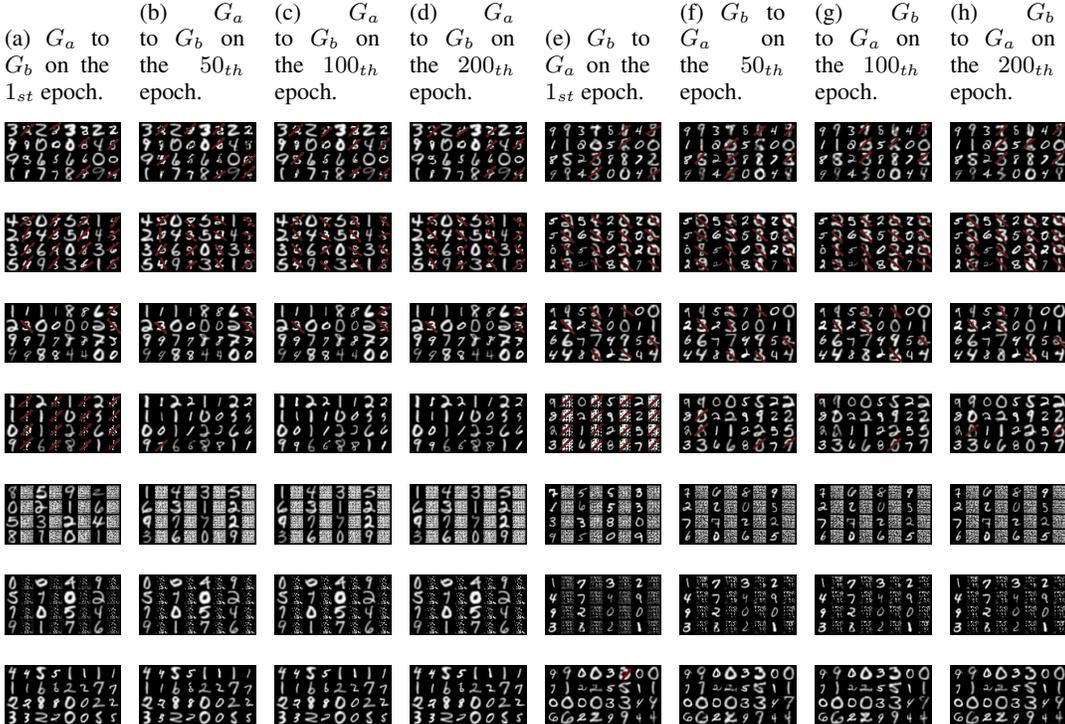


Figure 13: The generated samples from CycleGAN on corresponding optimisers (From top to bottom is SGD, SGDM, AdaM, PID, LPF-SGD, HPF-SGD and FuzzyPID).

Backward-Forward System do not utilise the "Steps per block", "Threshold", and "Positive samples portion" hyperparameters. Nonetheless, for a rigorous comparison, all experiments employed the same seven optimisers with consistent hyperparameters.

Table 4: Hyper-parameters for the image classification task on CIFAR10, CIFAR100 and TinyImageNet.

Hyper-parameter	VGG19	ResNet18	ResNet50	ResNet101	DenseNet121	MobileNetV2	EfficientNet
Data augmentation	Auto	Auto	Auto	Auto	Auto	Auto	Auto
Input resolution (CIFAR10,100)	[32,32,3]	[32,32,3]	[32,32,3]	[32,32,3]	[32,32,3]	[32,32,3]	[32,32,3]
Input resolution (TinyImageNet)	[64,64,3]	[64,64,3]	[64,64,3]	[64,64,3]	[64,64,3]	[64,64,3]	[64,64,3]
Epochs	200	200	200	200	200	200	200
Batch size	100	100	100	100	100	100	100
Hidden dropout	0	0	0	0	0	0	0
Random erasing prob	0	0	0	0	0	0	0
EMA decay	0	0	0	0	0	0	0
Cutmix α	0	0	0	0	0	0	0
Mixup α	0	0	0	0	0	0	0
Cutmix-Mixup	0	0	0	0	0	0	0
Label smoothing	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Peak learning rate	2e-2	2e-2	2e-2	2e-2	2e-2	2e-2	2e-2
optimiser	{SGD, SGDM, AdaM, PID, LPF-SGD, HPF-SGD, FuzzyPID}						

As shown in Table 4, we employed three datasets: CIFAR10, CIFAR100 and TinyImageNet. Additionally, one vision model – VGG19 which lacks the residual connection, and six residual connections used vision models are illustrated in our experiments. We specifically chose these seven vision models to investigate whether a more complex system, indicating a learning system order of two or higher, can be ascertained.

Meanwhile, we designed two filter processed SGD optimisers by using a second-order IIR structure. The coefficient of the convolution process in Equation 31 is listed in Table 5. Owing to the frequency cutoff around the midpoint (given the uncertainty in determining the sampling rate and the desired

Table 5: Coefficients of LPF-SGD and HPF-SGD using second-order IIR structure.

Filter Type	Gain	Numerator			Denominator		
	G	x_0	x_1	x_2	y_0	y_1	y_2
Low Pass Filter	0.49968	1	-0.99937	0.00063	1.0	0	-1.0
High Pass Filter	0.49968	1	0.99937	0.00063	1.0	0	-1.0

frequency band), this second-order IIR filter encompasses seven coefficients. It’s noteworthy that the **’filterDesigner’** toolbox in MATLAB can be utilized to design such filters.

F CIFAR10, CIFAR100 AND TINYIMAGENET

This section presents the accuracy rate using seven vision models (e.g., VGG19, ResNet18, ResNet50, ResNet101, DenseNet121, MobileNetV2 and EfficientNet) across seven optimisers (SGD, SGDM, AdaM, PID, LPF-SGD, HPF-SGD, and FuzzyPID). These results are detailed in Table 6, and the associated training and testing curves are depicted in Figure 14. VGG19 is a straight-forward connected vision model without residual blocks, and as demonstrated by the system response in Figure 10, no matter the assumed system order of VGG19 is one or two, compared to SGD on ResNet50, SGD on VGG19 only can achieve the half accuracy rate on CIFAR100 and TinyImageNet. Introducing a low pass filter to SGD results in a considerably slow learning curve ascent. Conversely, incorporating a high pass filter facilitates the learning process. We infer that the update of weights needs the high frequency component of gradient sequences to rapidly adapt to the optimal. Consistently, because of the adaptive part M in Equation 7, AdaM aims to follow the change of gradients with a faster speed. Nonetheless, relying solely on a single parameter, β_2 , for updates does not effectively mitigate the overshoot issue. Interestingly, the FuzzyPID optimiser exhibits a smoother learning trajectory compared to the PID. The design intention behind FuzzyPID was to supplement the PID optimiser, aiding in the adjustment of its overshoot issue. However, in practice, while FuzzyPID may not consistently outperform PID, it exhibits superior performance when deployed on CycleGAN.

Table 6: The results of CNN with different optimisers on CIFAR10 and CIFAR100. Using the 10-fold cross-validation, the average and standard variance results are shown below.

optimiser	SGD	SGDM	Adam	PID	LPF-SGD	HPF-SGD	FuzzyPID
CIFAR10							
<i>VGG19</i>	90.89 \pm 0.03	93.13 \pm 0.13	78.07 \pm 0.82	93.60 \pm 0.06	13.75 \pm 0.69	92.52 \pm 0.09	93.45 \pm 0.09
<i>ResNet18</i>	91.65 \pm 0.15	94.67 \pm 0.17	83.54 \pm 1.32	95.42 \pm 0.05	11.07 \pm 2.09	93.05 \pm 0.12	94.98 \pm 0.19
<i>ResNet50</i>	91.06 \pm 0.02	94.70 \pm 0.12	81.39 \pm 0.44	95.21 \pm 0.19	11.49 \pm 0.06	92.84 \pm 0.09	94.51 \pm 0.35
<i>ResNet101</i>	90.87 \pm 0.29	94.70 \pm 0.08	82.78 \pm 0.05	95.39 \pm 0.14	10.49 \pm 0.14	92.42 \pm 0.11	93.95 \pm 0.20
<i>DenseNet121</i>	91.37 \pm 0.15	95.20 \pm 0.28	84.62 \pm 0.27	95.71 \pm 0.04	11.90 \pm 0.69	93.23 \pm 0.07	94.53 \pm 0.13
<i>MobileNetV2</i>	87.97 \pm 0.07	93.78 \pm 0.01	83.02 \pm 0.24	94.12 \pm 0.25	10.02 \pm 0.75	90.74 \pm 0.12	93.46 \pm 0.03
<i>EfficientNet</i>	83.36 \pm 0.48	91.71 \pm 0.26	83.49 \pm 0.13	92.14 \pm 1.10	10.69 \pm 0.61	87.66 \pm 0.13	91.16 \pm 0.15
CIFAR100							
<i>VGG19</i>	66.42 \pm 0.28	71.97 \pm 0.21	18.28 \pm 2.12	73.25 \pm 0.05	1.35 \pm 0.05	69.65 \pm 0.21	73.21 \pm 0.29
<i>ResNet18</i>	70.05 \pm 0.03	76.03 \pm 0.02	49.90 \pm 0.02	77.84 \pm 0.01	1.12 \pm 0.02	72.54 \pm 0.17	76.78 \pm 0.08
<i>ResNet50</i>	66.52 \pm 0.42	77.25 \pm 0.16	49.64 \pm 0.80	78.37 \pm 0.04	1.28 \pm 0.10	71.23 \pm 0.10	75.62 \pm 0.16
<i>ResNet101</i>	64.69 \pm 0.11	76.36 \pm 0.38	50.97 \pm 0.91	79.25 \pm 0.26	1.09 \pm 0.04	70.61 \pm 0.14	72.78 \pm 0.31
<i>DenseNet121</i>	68.45 \pm 0.39	77.78 \pm 0.31	55.87 \pm 0.82	80.06 \pm 0.12	1.12 \pm 0.16	73.72 \pm 0.21	75.71 \pm 0.25
<i>MobileNetV2</i>	62.52 \pm 0.37	73.96 \pm 0.26	42.74 \pm 2.80	74.81 \pm 0.11	0.98 \pm 0.02	67.83 \pm 0.14	73.31 \pm 0.26
<i>EfficientNet</i>	50.35 \pm 0.77	66.87 \pm 0.38	34.06 \pm 7.41	71.32 \pm 0.29	1.01 \pm 0.04	55.98 \pm 0.21	62.92 \pm 0.35
TinyImageNet							
<i>VGG19</i>	44.87 \pm 0.01	51.22 \pm 0.20	0.57 \pm 0.07	0.50 \pm 0.00	0.00 \pm 0.00	46.66 \pm 0.07	0.50 \pm 0.00
<i>ResNet18</i>	50.27 \pm 0.25	58.79 \pm 0.15	34.33 \pm 1.23	63.71 \pm 0.71	0.00 \pm 0.09	54.46 \pm 0.09	61.39 \pm 0.13
<i>ResNet50</i>	42.20 \pm 0.08	63.54 \pm 0.20	35.01 \pm 1.18	67.51 \pm 0.40	0.00 \pm 0.00	48.38 \pm 0.10	60.82 \pm 0.50
<i>ResNet101</i>	40.98 \pm 0.31	64.13 \pm 0.19	35.64 \pm 0.71	69.54 \pm 0.56	0.40 \pm 0.00	48.22 \pm 0.06	58.77 \pm 0.21
<i>DenseNet121</i>	43.23 \pm 0.15	63.37 \pm 0.07	38.84 \pm 0.62	68.29 \pm 0.15	0.26 \pm 0.02	49.34 \pm 0.22	55.66 \pm 0.12
<i>MobileNetV2</i>	46.12 \pm 0.11	61.08 \pm 0.61	28.56 \pm 0.28	59.98 \pm 0.04	0.00 \pm 0.00	50.48 \pm 0.12	58.73 \pm 0.13
<i>EfficientNet</i>	54.12 \pm 0.22	62.28 \pm 0.04	23.11 \pm 0.03	62.66 \pm 0.12	0.30 \pm 0.02	57.58 \pm 0.04	60.43 \pm 0.39

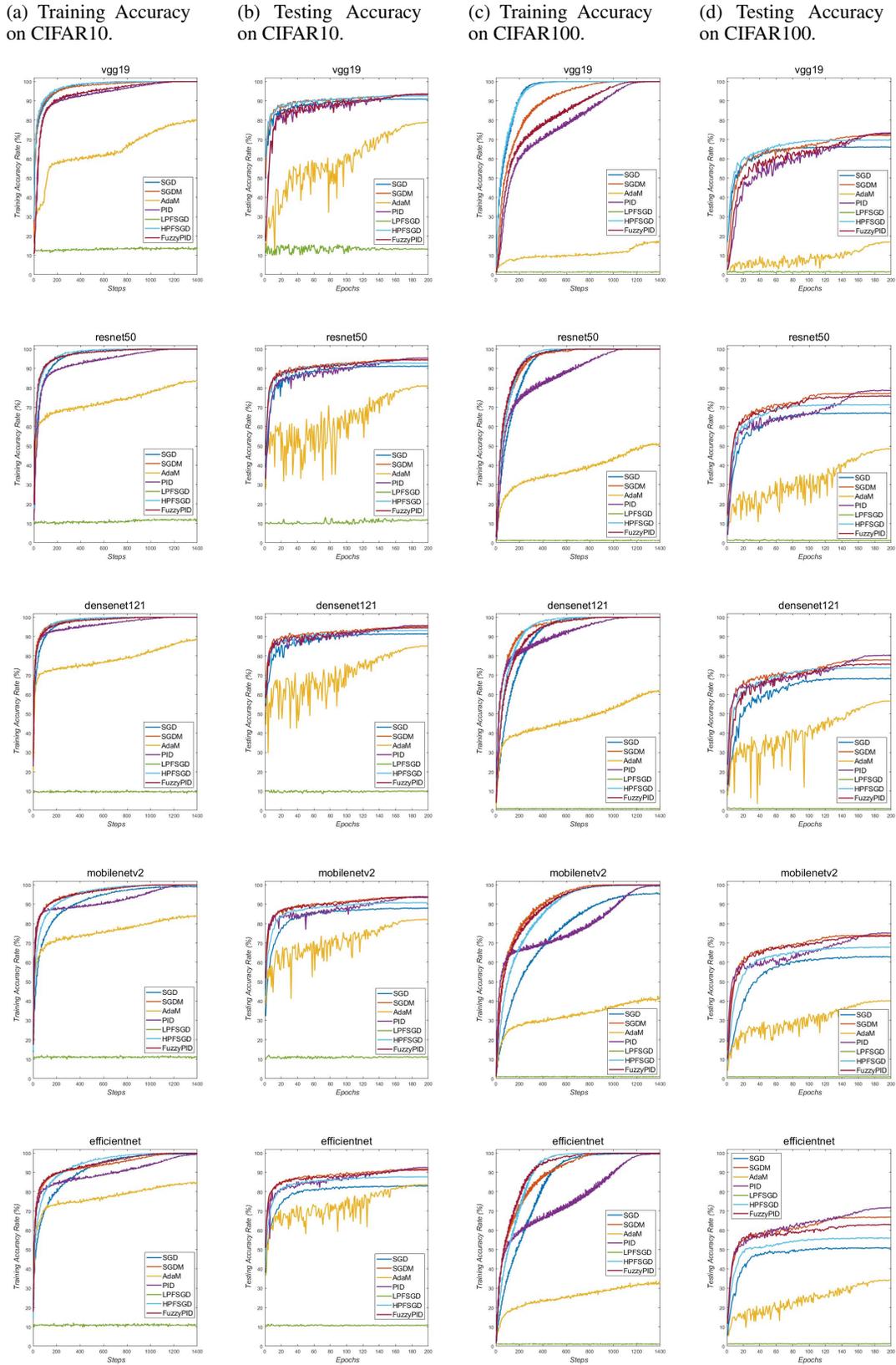


Figure 14: The training and testing curves of SOTA models on CIFAR10 and CIFAR100 datasets, and from the top to the bottom respectively is VGG19, ResNet50, DenseNet121, MobileNetV2 and EfficientNet on corresponding optimisers: SGD, SGDM, AdaM, PID, LPF-SGD, HPF-SGD and FuzzyPID.

(a) Training Accuracy on TinyImageNet200.

(b) Testing Accuracy on TinyImageNet200.

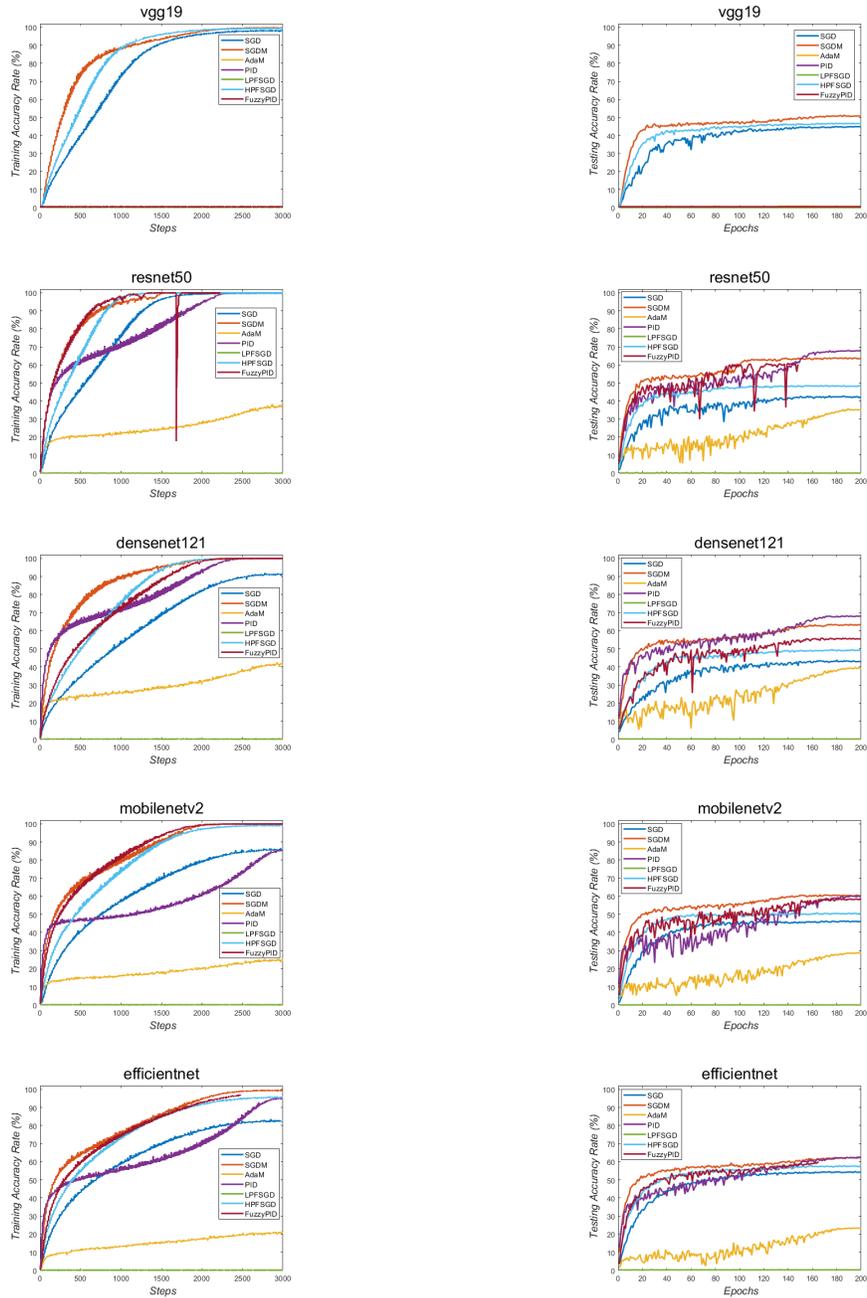


Figure 15: The training and testing curves of SOTA models (e.g., VGG19, ResNet50, DenseNet121, MobileNetV2 and EfficientNet) on TinyImageNet on corresponding optimisers: SGD, SGDM, AdaM, PID, LPF-SGD, HPF-SGD and FuzzyPID.