Controle PID em Redes Neurais Convolucionais

Gustavo F. G. Reis[1], **Milton E. R. Romero**[1]

[1]Faculdade de Computação (FACOM)
Universidade Federal do Mato Grosso do Sul (UFMS)

{gustavo_reis, milton.romero}@ufms.br

Resumo. *Este trabalho investiga a integração de controladores Proporcional-Integral-Derivativo (PID) em redes neurais convolucionais (CNNs) para aprimorar o desempenho de tarefas de classificação de imagens. Exploramos diferentes abordagens, incluindo a adição de uma camada com kernel PID, a substituição da primeira camada convolucional por um kernel PID e a aplicação de kernels PID em todas as camadas convolucionais. Os resultados obtidos indicam que a introdução de controladores PID nas CNNs pode influenciar significativamente o desempenho da rede, dependendo da configuração e dos parâmetros utilizados. Embora algumas configurações tenham apresentado resultados promissores, como um aumento na taxa de aprendizado, outras levaram a uma degradação do desempenho. Os resultados sugerem que a combinação de CNNs e PID's é uma área promissora para futuras pesquisas, com potencial para desenvolver modelos de aprendizado de máquina mais robustos e adaptáveis.*
Palavras-chave: *Redes neurais convolucionais, controlador PID, classificação de imagens, aprendizado de máquina.*

Abstract. *This paper investigates the integration of Proportional-Integral-Derivative (PID) controllers into Convolutional Neural Networks (CNNs) to enhance image classification performance. We explore different approaches, including adding a layer with a PID kernel, replacing the first convolutional layer with a PID kernel, and applying PID kernels to all convolutional layers. The results obtained indicate that introducing PID controllers into CNNs can significantly influence the network's performance, depending on the configuration and parameters used. While some configurations showed promising results, such as an increase in learning rate, others led to a degradation in performance. The results suggest that combining CNNs and PID's is a promising area for future research, with the potential to develop more robust and adaptable machine learning models.*
Keywords: *Convolutional neural networks, PID controller, image classification, machine learning.*

1. Introdução

A ideia de controlar sistemas automaticamente, ou seja, sem supervisão, iniciou-se com James Watt com o controle de velocidade de uma máquina a vapor no século XVIII, é um caso clássico da teoria de controle.

Alguns outros autores importantes para o desenvolvimento da teoria de controle são Minorsky que em 1922 configurou controladores automáticos de pilotagem de

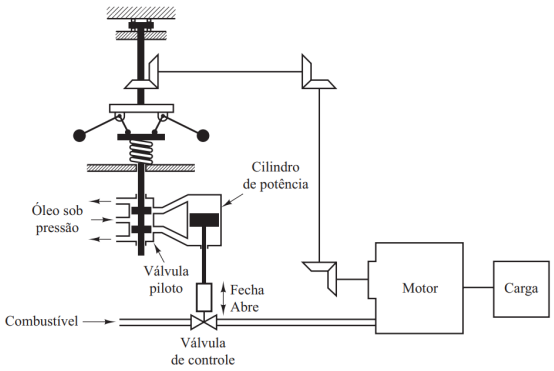


Figura 1. ***Controle da Máquina a Vapor: Conforme a rotação do motor aumenta a inércia das esferas fecha a válvula de combustível do motor que por sua vez tem a sua velocidade reduzida.***

embarcações mostrando a estabilidade que pudera ser determinada através de equações diferenciais [13]. No ano de 1932, Nyquist apresentou uma metodologia simples para a determinação da estabilidade em sistemas de malha fechada baseando-se na resposta de malhar aberta [14]. Em 1934 com Hazen introduzindo o termo "servomecanismos" para sistemas de controle de posição [15], ampliando a ideia de que sistemas podem ser controláveis. Na década de 40, alguns métodos de resposta em frequência como o diagrama de Bode [3], e também as regras criadas por Ziegler e Nichols para o ajuste de controladores PID (Proportional–integral–derivative), regras conhecidas como método de Ziegler-Nichols [18]. Já no final da década de 40 e começo da década de 50 o método de lugar das raízes de Evans [4] foi apresentando. A partir de 1960 no início da era digital, vários temas da teoria de controle foram debatidos, como exemplos o controle ótimo de sistemas e controle adaptativo e de aprendizagem, até chegarmos à teoria de controle moderno. Quando falamos sobre processamento e reconhecimento de imagens por aprendizado de máquina, uma alternativa é a CNN (Convolutional Neural Network), que leva dois conceitos principais em sua constituição, a convolução matemática e a ideia de rede neural. Em 1980 Fukushima apresentou o Neocognitron [6], um dos primeiros modelos de rede neural que explorava a ideia de camadas de convolução e pooling para reconhecimento de padrões. A LetNet-5 de 1998 desenvolvida por LeCun [11] foi uma das primeiras CNN's aplicadas com sucesso em problemas reais, ela foi utilizada para o reconhecimento de dígitos manuscritos. A progressão as CNN's tem permanecido, em 2012 a AlexNet [10] venceu uma competição de reconhecimento de larga escala de imagens (ILSVRC) com uma vantagem significativa das outras CNN's que participaram. Algumas das CNN's mais relevantes também, como a VGGNet desenvolvida no Visual Geometry Group da Universidade de Oxford em 2014 [16]. No mesmo ano a GoogLeNet (Inception) introduziu o conceito de módulos Inception [17], utilizando diferentes tamanhos de filtros em paralelo para capturar características em diferentes escalas. Em 2015 a ResNet proposta pela Microsoft Research [8], acrescentaram o conceito de conexões residuais as CNN's, que facilitou o treinamento de redes muito profundas. Pensando nos benefícios dessas duas áreas de estudo, existem vários estudos que mesclam as capacidades das CNN's na categorização de informações com as capacidades de PID's para controlar de maneira estável sistemas. Discutiremos na próxima seção alguns desses estudos relacionados.

2. Trabalhos Relacionados

Há algumas maneiras que relacionam CNN's e PID's, uma delas por X. Liu e Y. Dong em [12] que procuram substituir um controlador PID por um controlador CNN mapeando as entradas e saídas de um sistema, fazendo com que a automação do ajuste do controlador tivesse uma performance similar à técnica por PID. De forma diferente em [19] os autores desenvolveram o PIDNN que é um controlador PID onde seus parâmetros são atualizados por uma rede neural, e já que as redes neurais permitem uma flexibilidade maior durante o treinamento essa propriedade concilia o PID para que ele seja mais estável. No trabalho [5] os autores decidiram trabalhar com um controlador PID e um controlador CNN ao mesmo tempo para somar as tomadas de decisões de um robô seguidor de linhas, enquanto o controlador CNN reconhece o percurso e obstáculos o controlador PID decide os movimentos e a que velocidade o robô deve executá-las.

3. CNN e Controle PID

Ao definirmos um sistema de controle, definimos também o que são variáveis controladas e sinais de controle ou variáveis manipuladas. Uma variável controlada é uma grandeza ou a condição que é medida. Já o sinal de controle ou variável manipulada é grandeza que é modificada pelo controlador, desse modo a variável controlada tem o seu valor afetado. O conceito de uma planta é dado por um componente ou um conjunto de componentes de um equipamento funcionando de maneira integrada e realizam alguma operação. Sistemas definiremos por uma combinação de componentes que de forma conjunta trabalham para atingir um determinado objetivo. Essas definições podem ser vistas mais detalhadamente em [15]. Num controlador PID nós temos as três grandezas que são relacionadas ao (erro) do nosso sistema, o erro é dado pela diferença da saída do sistema com o valor desejado, por exemplo na figura 1 o valor da rotação desejada do motor em diferença com o valor atual. As três grandezas do PID são Proporcional(P) quando a saída do controlador é proporcional ao erro atual, se o erro for pequeno a compensação também será pequena e vice-versa, Integral(I) a saída do controlador é proporcional à integral do erro ao longo do tempo, conforme o tempo passa o integral ajusta o controle com a soma dos erros, e Derivativo(D) onde a saída é relativa a derivada do erro, relativa a taxa do erro, isso significa que a parcela derivativa tenta predizer o erro antes que ele aconteça [2].

Para falar sobre as redes neurais convolucionais (CNN's) começaremos pelas condições matemáticas que originaram esse modelo de rede neural, a primeira sendo a convolução que dá origem ao nome dessa rede, e a segunda função de pooling que está presente na maioria das CNN's. De maneira generalizada, a convolução é uma operação sobre duas funções sobre valores reais. Imagine uma função $x(t)$, a cada instante t real podemos medir um valor de $x(t)$, queremos aqui aplicar uma média durante um intervalo em t, mas o valor atual de t deve ser mais relevante que os valores anteriores. Para isso aplicamos uma função de ponderação $w(a)$, que multiplicada pela nossa função x encontramos a suavização

$$s(t) = \int x(a)w(t-a)da. \tag{1}$$

Essa operação é denominada convolução, que também é representada com um asterisco entre duas funções

$$s(t) = (x * w)(t). \tag{2}$$

Utilizamos convolução em CNN's pois há três importantes ideias que melhoram o aprendizado, interações esparsas, compartilhamento de parâmetros e representação equivariantes, sem contar que a convolução possibilita trabalhar com informações de entrada de tamanho variável [7].

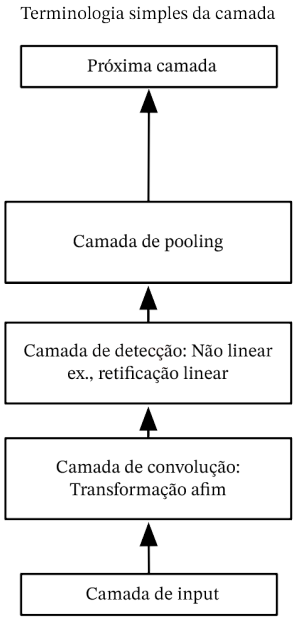


Figura 2. ***Estrutura básica de uma camada CNN.***

Uma camada de uma CNN é dada assim como mostra a figura 2, o estágio de convolução, o estágio de de ativação e o estágio de de pooling. O estágio de convolução é responsável por gerar ativações lineares, essa ativações no segundo estágio são transformadas por uma função de ativação linear e por fim a informação chega ao estágio de pooling. A função de pooling é responsável por traduzir uma característica detectada a um valor estatístico, assim ao final da camada o valor daquela característica tende a ser o mesmo independentemente da translação que ele possa sofrer na entrada, a função pooling permite a invariância de características a posição em que elas se encontram [7].

4. Desenvolvimento

Assim como na seção 2 vimos controladores CNN e PID trabalhando em conjunto para otimizar realização de tarefas, também vimos uma CNN atualizando os parâmetros de um controlador PID buscando otimizar o seu desempenho aceitando uma certa flexibilidade a variação dos dados trabalhados fazendo com que o PID seja mais consistente. A ideia aqui é verificar os impactos que um controlador PID tem em uma rede neural convolucional.

Na arquitetura da estratégia é colocado um filtro PID na camada convolucional, esse filtro é atualizado de acordo com a precisão da CNN. O objetivo da nossa CNN é classificar as imagens do conjunto de dados CIFAR-10 [9], o conjunto de dados consiste em 60.000 imagens coloridas de 32x32 pixels em 10 classes, com 6.000 imagens por

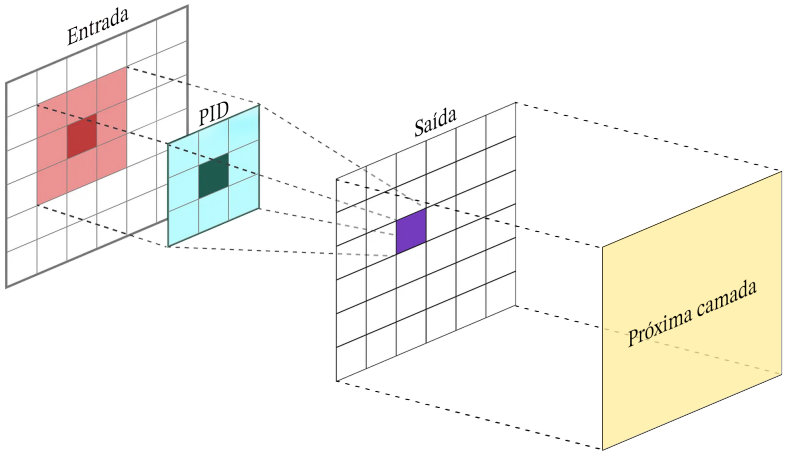


Figura 3. ***Arquitetura da aplicação com kernel PID.***

classe. Existem 50.000 imagens de treinamento e 10.000 imagens de teste, as 10 classes são avião, automóvel, pássaro, gato, cervo, cachorro, sapo, cavalo, navio e caminhão.

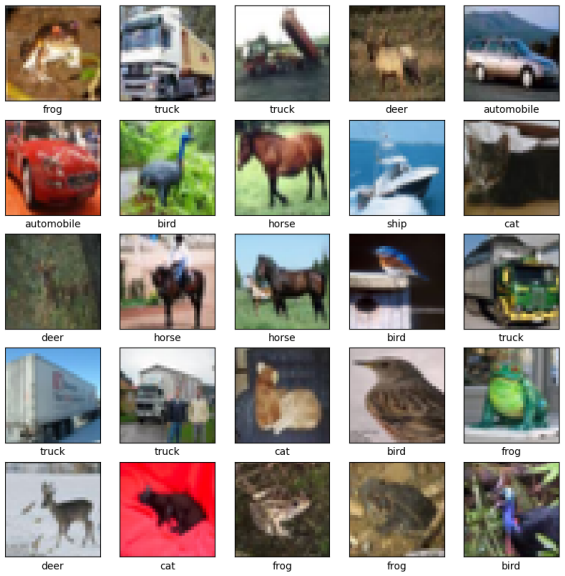


Figura 4. ***Exemplos aleatórios do dataset CIFAR-10***

Para essa tarefa escolhemos como base uma CNN simples baseada na API Keras do TensorFlow [1] que possui três camadas convolucionais, e duas camadas de pooling, como mostra a figura 5.

Escolhemos essa configuração para a CNN pela simplicidade, assim implementar alterações e observar os impactos diretos e indiretos também se torna simples, evitando falsas impressões da implementação de um kernel PID no nosso caso. Esse filtro, que é implementado como um kernel para a CNN, é constituído pela combinação de três matrizes 3x3, e cada uma dessas matrizes é multiplicada pelo termo proporcional K_p, integral K_i e derivativo K_d. Pensando nos conceitos do controlador proporcional, ele implica a compensação pelo erro atual, colocando isso em uma convolução entre matrizes, o cálculo atual é dado sobre o elemento central, assim o termo proporcional matricial do

Layer (type)	Output Shape	Param #
conv2d_70 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_42 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_71 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_43 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_72 (Conv2D)	(None, 4, 4, 64)	36,928
flatten_16 (Flatten)	(None, 1024)	0
dense_32 (Dense)	(None, 64)	65,600
dense_33 (Dense)	(None, 10)	650

Total params: 122,570 (478.79 KB)

Trainable params: 122,570 (478.79 KB)

Non-trainable params: 0 (0.00 B)

Figura 5. ***Modelo CNN base***

nosso kernel é:

$$K_p \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

e o no termo integral, queremos compensar sobre os erros passados, então consideramos todos os pixels da matriz para serem controlados, com o termo integral matricial sendo:

$$K_i \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

já no termo derivativo, queremos compensar o erro predito, então fazemos a diferença entre os extremos da matriz do termo derivativo, ficando:

$$K_d \cdot \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

a soma de todos esses termos matriciais é o nosso kernel PID, dado por:

$$\text{Kernel PID} = K_p \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + K_i \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} + K_d \cdot \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

$$\text{Kernel PID} = \begin{bmatrix} K_i - K_d & K_i - K_d & K_i - K_d \\ K_i & K_p + K_i & K_i \\ K_i + K_d & K_i + K_d & K_i + K_d \end{bmatrix}.$$

5. Experimentos e Resultados

Para poder ajustar os parâmetros do nosso kernel PID, nós calculamos o erro em cima de três grandezas durante a experimentação, utilizamos os valores de accuracy, val_accuracy e loss dados durante o treino da CNN. Então o erro é dado pela diferença entre esses valores e seu valor ideal, para a accuracy e a val_accuracy o valor ideal é 1 representando 100% de acerto, e para loss o valor ideal é 0 representando nenhuma perda.

Layer (type)	Output Shape	Param #
conv2d_60 (Conv2D)	(None, 30, 30, 1)	28
conv2d_61 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_36 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_62 (Conv2D)	(None, 12, 12, 64)	18,496
max_pooling2d_37 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_63 (Conv2D)	(None, 4, 4, 64)	36,928
flatten_13 (Flatten)	(None, 1024)	0
dense_26 (Dense)	(None, 64)	65,600
dense_27 (Dense)	(None, 10)	650

Total params: 122,022 (476.65 KB)

Trainable params: 122,022 (476.65 KB)

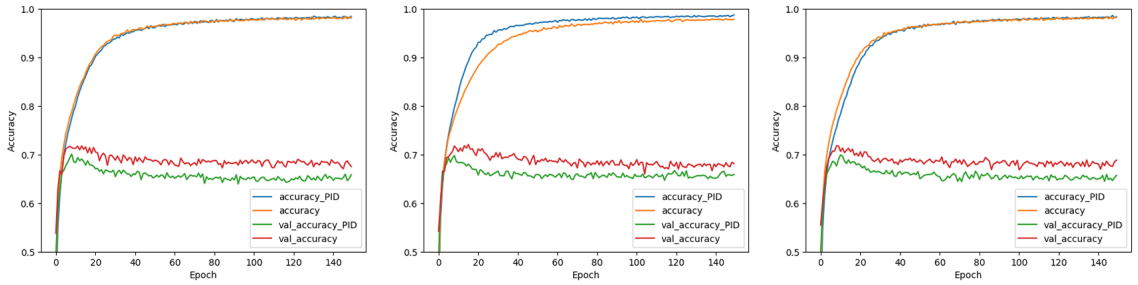
Non-trainable params: 0 (0.00 B)

Figura 6. ***Modelo CNN adicionando uma camada com kernel PID.***

5.1. CNN adicionando uma camada com kernel PID

Apenas acrescentando uma camada a mais com o nosso kernel PID, a configuração da CNN ficou como mostra a figura 7,

e utilizando a CNN base como comparação, para os parâmetros do PID sendo ajustados para accuracy, val_accuracy e loss os valores encontrados foram:



(a) accuracy (b) loss (c) val_accuracy

Figura 7. ***Gráficos resultantes de accuracy e val_accuracy da CNN adicionando uma camada com kernel PID.***

Em todas as referências, o val_loss esteve abaixo da CNN base, mas, usando loss como referência, o valor da accuracy durante o treino da CNN com a primeira camada PID cresceu significativamente mais rápido e se manteve acima da CNN base.

5.2. CNN substituindo a primeira camada com kernel PID

Substituindo a primeira camada convolucional pela com o nosso kernel PID, a configuração da CNN ficou como mostra a figura 8,

importante notar que a quantidade de parâmetros é menor pois a camada com o kernel PID tem menos parâmetros que a convolucional base, comparando novamente com os parâmetros do PID sendo ajustados para accuracy, val_accuracy e loss, obtivemos:

Neste experimento, utilizando todas as referências, a influência do PID foi negativa, resultado que pode ser justificado pela remoção de uma camada em relação ao modelo de CNN, que apenas foi acrescentada uma camada com o kernel PID.

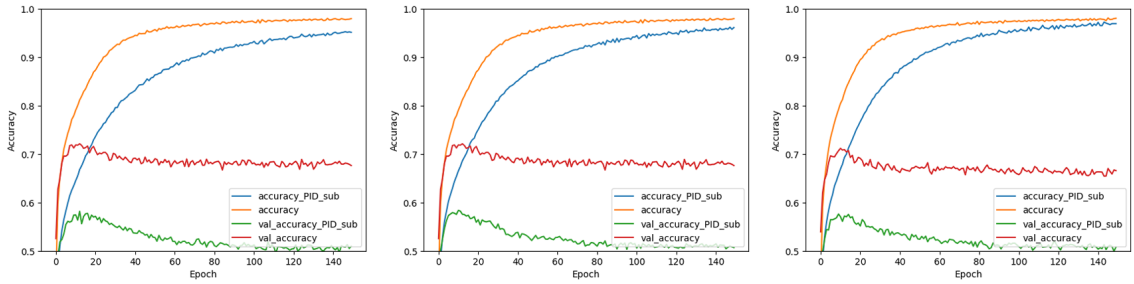
Layer (type)	Output Shape	Param #
conv2d_64 (Conv2D)	(None, 30, 30, 1)	28
max_pooling2d_38 (MaxPooling2D)	(None, 15, 15, 1)	0
conv2d_65 (Conv2D)	(None, 13, 13, 64)	640
max_pooling2d_39 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_66 (Conv2D)	(None, 4, 4, 64)	36,928
flatten_14 (Flatten)	(None, 1024)	0
dense_28 (Dense)	(None, 64)	65,600
dense_29 (Dense)	(None, 10)	650

Total params: 103,846 (405.65 KB)

Trainable params: 103,846 (405.65 KB)

Non-trainable params: 0 (0.00 B)

Figura 8. ***Modelo CNN substituindo a primeira camada por uma camada com kernel PID.***



(a) accuracy (b) loss (c) val_accuracy

Figura 9. ***Gráficos resultantes de accuracy e val_accuracy da CNN substituindo a primeira camada por uma camada com kernel PID.***

5.3. CNN com todas as com kernel PID

Nessa configuração, substituímos todas as camadas convolucionais pelas com o nosso kernel PID, porém aqui em cada camada das três havia um filtro PID diferente, sendo ajustado na primeira camada pela accuracy, a segunda pela loss e a terceira pela val_accuracy. E também ajustamos a quantidade de parâmetros das camadas para os mesmos valores das camadas da CNN base.

Layer (type)	Output Shape	Param #
conv2d_67 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_40 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_68 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_41 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_69 (Conv2D)	(None, 4, 4, 64)	36,928
flatten_15 (Flatten)	(None, 1024)	0
dense_30 (Dense)	(None, 64)	65,600
dense_31 (Dense)	(None, 10)	650

Total params: 122,570 (478.79 KB)

Trainable params: 122,570 (478.79 KB)

Non-trainable params: 0 (0.00 B)

Figura 10. ***Modelo CNN com todas as camadas com kernel PID.***

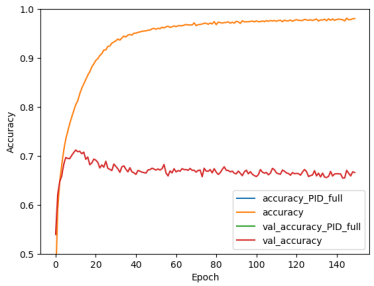


Figura 11. ***Gráficos resultantes de accuracy e val_accuracy da CNN com todas as camadas com kernel PID.***

Neste último experimento, a CNN com todas as camadas com um kernel PID se perdeu completamente, com o PID tentando compensar os erros a cada camada e removendo as propriedades das informações durante o treinamento.

6. Conclusão

Os resultados mostraram que a influência foi negativa na forma como o PID foi implementado na CNN que escolhemos como base, porém na seção 5.1 quando utilizado o loss como referência, durante o treino a accuracy mostrou uma reação positiva em relação à performance, o que pode indicar que reajustando parâmetros, revendo a implementação matemática e referencial do PID para a CNN, nós podemos adquirir resultados distintos e, quem sabe, resultados com melhor performance. Podemos concluir que fica em aberto para futuras tentativas seguir modelando estratégias para utilizarmos a robustez dos PID para melhorar a assertividade das redes neurais convolucionais.

Referências

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] R. C. D. R. H. Bishop. *Modern control systems*. Addison-Wesley, 2011.

[3] H. W. Bode. Network analysis and feedback amplifier design. *(No Title)*, 1945.

[4] W. R. Evans. Control system synthesis by root locus method. *Transactions of the American Institute of Electrical Engineers*, 69(1):66–69, 1950.

[5] R. Farkh, M. T. Quasim, K. Al Jaloud, S. Alhuwaimel, and S. T. Siddiqui. Computer vision-control-based cnn-pid for mobile robot. *Computers, Materials & Continua*, 68(1), 2021.

[6] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[7] I. Goodfellow. Deep learning, 2016.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www.cs.toronto.edu/ kriz/cifar.html*, 6(1):1, 2009.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[11] Y. LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.

[12] X. Liu and Y. Dong. A convolutional neural networks approach to devise controller. In *MATEC Web of Conferences*, volume 139, page 00168. EDP Sciences, 2017.

[13] N. Minorsky and T. Teichmann. Nonlinear oscillations. *Physics Today*, 15(9):63–65, 1962.

[14] H. Nyquist. Regeneration theory. *Bell system technical journal*, 11(1):126–147, 1932.

[15] K. Ogata et al. *Modern control engineering*. Prentice Hall India, 2009.

[16] K. Simonyan. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014.

[18] J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. *Transactions of the American society of mechanical engineers*, 64(8):759–765, 1942.

[19] A. Zribi, M. Chtourou, and M. Djemel. A new pid neural network controller design for nonlinear processes. *Journal of Circuits, Systems and Computers*, 27(04):1850065, 2018.

A. Código e Implementação

Disponibilidade do Código: O código desenvolvido no formato Python Notebooks para a execução desses experimentos pode ser encontrado em `https://github.com/gusta-reis/CNN_plus_PID`

Conflitos de Interesse Financeiro: Declaramos nenhum interesse conflituoso tanto para a implementação desses experimentos quanto para a ideia.

B. Experimentos Avulsos

No período inicial das experimentações, fizemos algumas aquisições, algumas delas que estavam fora do escopo definido para este trabalho.

No gráfico da figura 12 embora o comportamento seja parecido com os experimentos desenvolvidos neste trabalho, aqui o kernel PID não estava atualizando os seus valores (anterior + atualizações), e sim a cada vez que a função de atualização era chamada, ela sobrescrevia o kernel antigo eliminando toda a informação adquirida por ele anteriormente. O problema de lógica foi corrigido posteriormente na função de atualização do PID.

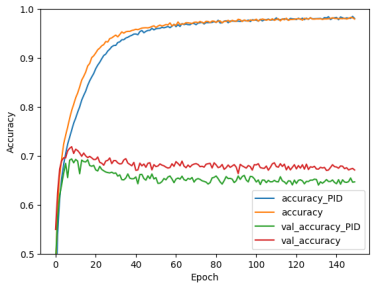


Figura 12. ***Gráficos resultantes de accuracy e val_accuracy da CNN adicionando o filtro PID, porém o filtro é sobrescrito a cada atualização.***

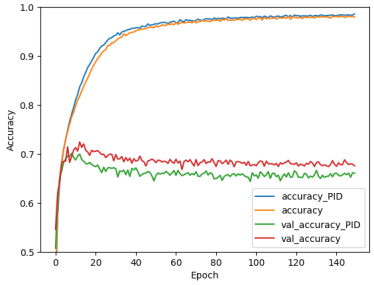


Figura 13. ***Gráficos resultantes de accuracy e val_accuracy da CNN adicionando o filtro PID, porém o valor referência para o erro do proporcional e integral estavam diferentes.***

Na figura 13 o cálculo de erro ainda não havia sido unificado para os termos proporcional, integral e derivativo, assim na hora de reajustar o filtro, erros diferentes eram calculados para atualizar um único filtro.